MOLECULAR ARTICULATION
IN RESPONSE TO INTERACTIVE ATOMIC FORCES
IN DOCKER

THESIS

Todd R. Kellett, Captain, USAF

AFIT/GCS/ENG/96D-15

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/96D-15

MOLECULAR ARTICULATION IN RESPONSE TO INTERACTIVE ATOMIC FORCES IN DOCKER

THESIS

Todd R. Kellett, Captain, USAF

AFIT/GCS/ENG/96D-15

AFIT/GCS/ENG/96D-15

MOLECULAR ARTICULATION IN RESPONSE TO INTERACTIVE ATOMIC FORCES IN DOCKER

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air Education and Training Command

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Science

Todd R. Kellett, B. S.

Captain, USAF

December 1996

Approved for public release; distribution unlimited

Acknowledgments

I am indebted to my sponsor, Dr. James Lupo. His guidance and wealth of knowledge were invaluable in the accomplishment of this thesis. He was never to busy for a question and he helped to open several doors for me, both metaphorically and physically.

I would also like to thank my thesis advisor, Maj. Keith Shomper. His guidance was essential to my thesis and helped to shape the final product. Hopefully this will be the last time you have to read this document!

Finally, I would like to thank my wife, Jenny. She provided support throughout this long and arduous process. She provided encouragement and motivation through the late nights and weekends that were spent at AFIT working on thesis and class work. Her support helped carry me through the thesis and AFIT program.

Todd R. Kellett

Table of Contents

Page

Page

List of Figures

Page

List of Tables

Page

Abstract

Molecular docking aids in the design of materials supporting the current and future needs of the warfighter by simulating the real-world results of combining molecules. Specifically, it supports research and development in novel non-linear optical materials for laser-hardening and other advanced optical applications. Potential uses for these materials range from personnel and optical system laser protection to holographic information displays. DOCKER, our baseline docking system, minimizes the computational overhead of the simulation by modeling the molecules as rigid objects. This simplification can cause DOCKER's solutions to disagree with the real-world results, because molecules flex as they react to one another and to other external influences, such as heat or the presence of a solvent. Our research adds flexibility to the DOCKER model by articulating it in response to the interactive atomic forces. We anticipate this addition will improve the model's predictive capability by improving its overall fidelity. Articulation also provides a basis for other extensions. These extensions include thermal effects and computation of barrier energy along a reaction path.

AFIT/GCS/ENG/96D-15

Molecular Articulation in Response to Interactive Atomic Forces in DOCKER

1. Introduction

Man's desire to create has lead to the discovery of many new materials throughout history. Initial creation of materials resulted from either accident or blind experimentation where the chemist had no educated guess of the results. This experimentation provided the basis of material science by identifying combinations of chemicals that created new materials and combinations to be avoided. As time progressed, material science matured. Experimentally combining chemicals and materials has provided chemists with the information necessary to create specific interactions. The compilation of this knowledge over the years has improved the capabilities of the chemist by providing the basic building blocks for creation of more complex materials. However, the creation of new materials still necessarily involves experimentation. Technology has caught up with science, and tools have been developed to provide easy access to this compilation of information, assisting the chemist with new material development. These tools include models that define the interaction of molecules at the atomic level, and tools which use these models to manipulate molecules.

DOCKER is a molecular docking tool which simulates the combination of two molecules to form a new material. Simulation has provided a method for chemists to test the resultant properties of a combination of molecules. These results determine if the combination will meet the needs of the material they are researching without actually having to perform a physical experiment. DOCKER allows the chemist to test different orientations between the molecules by manipulating one of them. The overall results of applying DOCKER to the material development field are a savings in materials and time. Experiments that will fail are identified in a simulation before the performance of any physical experiment that will waste materials. Identifying faulty experiments early also saves time. The chemist will not have to perform any analysis of the results of the experiment to determine that it was a failure as the simulation will have already provided that information.

DOCKER consists of a main program called *dock* and three servers: the display, energy, and arm servers. These four individual processes communicate through message passing. *Dock* drives the entire system and controls the three servers. The display server handles the graphics necessary to display the molecules. The display is currently limited to a bond vector diagram. The energy server does the calculations to determine the total energy of the molecules in their current orientation. The arm server communicates with the manipulator arm. This arm is an input device for positioning the drug molecule and an output device for sensing the force of the attraction or repulsion between the molecules.

1.1 Problem Description

The main problem with DOCKER is a lack of realism. The simulation provides information on the interaction of the molecules, but the information is not realistic because the molecules are modeled as rigid objects. In the real world, molecules bend and move when interacting with other molecules. Allowing flexibility in the molecules will move the simulation closer to results that will match the real-world interactions. Implementing articulation in response to the interactive atomic forces within the molecules achieves this flexibility. Articulation involves allowing each atom in the system to react to the forces applied to it by the other atoms. The resulting motion will move the molecules to a low energy position, similar to a system of springs finding a resting position, defining the actual results of docking the molecules in that orientation. The goal of this thesis is to implement articulation in response to interactive atomic forces in DOCKER.

A secondary problem with DOCKER is the lack of a tool to orient the molecules before the docking simulation. The current method for orienting the molecules is a guess by the user which only allows translation of the molecules and provides no visual feedback. This guess wastes time because it requires the user to verify it, by loading it in DOCKER, and inevitably correct it. The second goal of this thesis is to build an orientation tool that will allow the user to translate and rotate the molecules using a visual display to put the molecules into an acceptable orientation for the docking simulation.

1.2 Assumptions

Several assumptions are required to formulate my initial approach to the problem. These assumptions are:

1. Wright Laboratories will provide access to equipment necessary to run DOCKER. The program requires the use of the PER-Force arm and specific machines to run the display server, energy server and arm server. Implementing and testing my implementation of articulation will require the use of this equipment.
2. The Virtual World will support the modification of individual objects in the display. Articulation will require the ability to modify the location of each vector in the display which will represent the change in the location of the endpoint atoms.

3. The network can handle the increased traffic between the servers. Articulation will increase the number of messages passed between the energy server and dock, and dock and the display server.

4. Motif can support the molecular manipulation desired by the user. Motif does not provide any manipulation of objects directly, but it does provide tools to build the desired manipulation tools.

1.3 Scope

Articulation in response to interactive forces within the molecules affects the entire DOCKER system. It involves changing the display system and the energy calculation system. This project will focus only on the display system. Additionally, the project only provides articulation at the user's request and not during user manipulation of the drug molecule. The purpose of this project is to provide a first step toward the implementation of articulation. This project will demonstrate the ability of the display system to support the changes and the increased load of information required for articulation.

The orientation tool will only implement the functions to orient the molecules. These functions will include the ability to translate and rotate the molecules, rotate the world, and dolly the users eye closer to or farther away from the molecules. The tool will also provide functions to display the extent of the molecules along each axis, load molecules, and save the molecules after orientation. Functions not included in the scope of this project are rotation of a molecule about its center of mass, allowing the user to scoop a spherical region out of a molecule, and make a planar cut through a molecule.

1.4 Approach Overview

1.4.1 Articulation in Response to Interactive Atomic Forces

Articulation involves three steps: getting the new coordinates for each atom from the energy server, calculating the changes to the display, and updating the display with the changes. As the energy calculations are out of the scope of this project, the energy server is implemented as a stub that returns predetermined sets of coordinates. *Dock* calculates the changes to each vector in the display based on the new coordinates for each atom and builds a message for the display server. The display server accepts this message and changes the vectors being displayed. These three steps are repeated to create a series of articulation steps that make the molecules flex in the display.

1.4.2 Orientation Tool

The orientation tool is designed in Motif. It provides a viewing area to display the molecules and allows the user to manipulate them with the mouse. It has a standard windowing system menu bar to make it easier for the user to learn and use. The tool also provides a numerical display of the extent of the molecules along each axis.

1.5 Users' Desired Progression

1.5.1 Articulation in Response to Interactive Atomic Forces

The users of DOCKER have expressed a desire to enhance it with new functionality. The first step in this enhancement process is the implementation of articulation at the users request, which is the focus of this project. The next step is to implement the energy calculations that are necessary to support articulation. These calculations will be easier to implement if the model used to calculate the energy is changed from the AMBER model to CHARMm. The final step is to implement real-time articulation in DOCKER allowing the molecules to flex in response to molecular forces at all times.

1.5.2 Display Server

DOCKER currently displays the molecules using a bond vector drawing. The users have expressed a desire to be able to display the molecules with other drawing techniques such as a ball and stick drawing or using various surface rendering techniques. Another area of interest is holographic display. The ability to display the molecules as 3D holograms opens up a new area of molecular viewing and manipulation techniques.

1.5.3 Model Construction Tool

The users desire a model construction tool to set up the molecules for DOCKER. This tool will perform basic orientation functions such as rotation, translation, and zooming the view of molecules. The orientation tool designed for this project performs these functions. Additional desires for the model construction tool are to rotate a molecule about its center of mass, scoop a spherical region out of a molecule, and making a planar cut through a molecule. Having a single tool to perform all of these functions will simplify the setup for a docking simulation.

1.6 Thesis Overview

This thesis is divided into six chapters. Chapter two presents background information necessary to understand molecular visualization and docking. It also discusses the history of DOCKER and how it currently operates. Chapter three discusses the requirements placed on me by the users and the approach that I took to solve the problem. The fourth chapter covers the design and implementation of my solution. Chapter five presents the results of my implementation. The final chapter discusses future work to support and enhance articulation in DOCKER.

2. Background

2.1 Molecular Visualization

Molecular visualization provides the ability for the chemist to see what he/she is unable to see with the naked eye. It provides a graphical representation of a molecule on an atomic level to allow the chemist to see the structure and interactions of the atoms. In addition to being able to see the molecule, molecular visualization has given chemists the ability to comprehend complex sets of information about the molecules [BRYSON96]. The chemist can see different atoms, bonds, and energy present in the system and understand how they are all interacting.

Chemist Vivian Cody, a senior research scientist and head of structure and drug design at the Medical Foundation of Buffalo, has used molecular visualization provided by a virtual environment designed by the University of North Carolina, Chapel Hill, to study the docking of drug molecules in protein receptors. This technology has allowed her to discover that her interpretation of the folding pattern of a human protein called transthyretin was incorrect. She was able to see the folding pattern with the molecular visualization tool and gain a better understanding of the molecule [ILLMAN94].

2.1.1 Sculpt

Mark Surles, a former student at University of North Carolina, Chapel Hill, has been continuing his Ph.D. work at the San Diego Supercomputer Center with a molecular modeling system called Sculpt [SURLES94]. The Sculpt system is a molecular visualization system designed to allow a chemist to interact with a molecule by tugging an atom within it. The model then adjusts the remaining atoms in the molecule until it reaches a minimum energy orientation, much like a series of balls attached by springs would eventually reach a minimum energy orientation and stop bouncing. The system was designed to be used on protein molecules and takes advantage of the protein structure to simplify the calculations that must be performed.

2.1.2 GAME

Another example of a molecular visualization tool is the GAME (Graphic Adjusted Monopole Expansion) algorithm created by students at the Universität Stuttgart in Stuttgart, Germany [HAYD95]. This algorithm is a new method for calculating and displaying the electrostatic potential of a molecule. The students accomplished this by defining a surface which encompasses the molecule and representing the electrostatic potential with color codes at user defined points on the surface. They demonstrated that this representation is useful in determining how a drug molecule will interact with a protein molecule during a docking task.

2.1.3 Material Research

At the Tohoku University in Japan, researchers have studied the formation of ultrafine gold particles on an MgO(100) surface using molecular visualization [KUBO95]. They were able to show at low temperatures, gold particles dropped on the MgO surface were drawn together to form clusters. Furthermore, they were able to show the cluster would form at the location of a point defect in the surface of the MgO surface. Controlling the placement of the point defects allowed the researchers to control how the clusters formed. This discovery is significant because it could be used to design atomically controlled thin films and catalysts.

2.1.4 Computing Smooth Molecular Surfaces

Students at the University of North Carolina, Chapel Hill, have developed a method for computing and visualizing smooth molecular surfaces [VARSHNEY94]. A smooth molecular surface of a molecule is defined as the surface an exterior probe sphere would touch as it rolled over the atoms of the molecule. Their method is significant because it parallelizes easily and scales linearly with the number of atoms in the molecule making it very efficient. Their method is also useful to chemist because it visualizes the changes that occur within a molecule during manipulation of the molecule as in molecular docking.

2.2 Molecular Docking

In addition to visualizing molecules, chemists have been interested in the process of combining molecules to create new materials, medicines, and chemicals. This process, known as molecular docking, consists of determining the best orientation for attaching two molecules to form a single molecule. By selecting molecules with certain properties, chemists can create a molecule with a desirable combination of properties of the original molecules.

When discussing molecular docking, chemists refer to the two molecules as a protein and a drug. The original focus of molecular docking, and still the major focus to date, is the medical field. In this field, molecular docking is used to determine ways to dock drug molecules with actual or synthetic protein molecules. The goal is to find a drug that would inhibit, prevent, or modify some specific reaction in the protein molecule, thereby producing a cure or other therapeutic effect. An example of this application is cancer research in which molecular docking is used to find new binding orientations for anticancer drugs with human enzymes which are chemotherapy targets [ILLMAN94].

Molecular docking in the materials research field has adopted the terms protein and drug to refer to the molecules which they attempt to dock. The protein (also known as the receptor) refers to the static molecule which the drug molecule is being attached to. The drug molecule is usually the smaller of the two molecules and is manipulated by the chemist to probe the protein to locate a desirable docking position.

There are several methods the chemist can use to determine the best or desirable docking position of two molecules. The first is the total interaction energy in the system. The interaction energy in the system represents the net attraction or repulsion between the molecules. The lower the total energy is, the more attractive and stable the docking position will be.

The second method is to use an analysis tool after the docking position has been determined. This will provide the chemist with information about the final position of each atom in the system and several other qualities of the molecule including its energy level and stability.

2.2.1 Water Purification Study

Researchers at the Tohoku University in Japan have been using molecular dynamics and molecular docking techniques to study methods for removing NO_x from a water supply [HIMEI95]. Their research focused on Ga-ZSM-5 and its role in the process of removing NO_x from water. They used molecular docking and quantum chemistry to determine how the molecules would combine and based their conclusions on the results.

2.2.2 Cancer Research

The Environmental Protection Agency (EPA) has announced that they will require molecular data on how chemicals cause cancer [STONE95]. The EPA is driving a shift from the method of giving an animal high doses of a chemical to determine its cancer risk, which can provide misleading information, to demonstrating how the chemical will cause cancer. The molecular data requested by the EPA will require chemist and biologists to use molecular docking and molecular dynamics tools to show at the atomic level how the chemical interacts with the body of the host causing the cancer.

2.3 Molecular Research at the University of North Carolina, Chapel Hill

The University of North Carolina, Chapel Hill, has been working in molecular docking since 1967. The work at UNC was inspired by the vision of Ivan Sutherland. In 1965 Ivan Sutherland set forth his vision of "The Ultimate Display" as a window into a virtual world that incorporated visual, auditory, and haptic interaction [SUTHERLAND65, BROOKS90]. Haptic refers to the sense of touch as well as the sense of where one's limbs are in space.

GRIP was a molecular fitting system developed at UNC [BRITTON77]. The system allowed the chemist to adjust individual atoms in a molecule to an optimum fit against electron density distributions [BROOKS88]. The GRIP system was originally designed as a pharmaceutical application [BENNETT92] and, according to its users, was the first to solve a new protein entirely with virtual models.

The Raster Molecule Docking System allowed the chemist to dock a movable drug molecule to a static protein molecule [PALMER87]. The drug was represented as a thin stick figure and the protein was represented by spheres with thin cavities that the drug could be moved into [BROOKS88]. This system also provided collision detection for the user.

Project GROPE was another molecular docking system developed at UNC [BATTER71, KILPATRICK76, OUH-YOUNG90]. In this system, the chemist maneuvered the drug molecule with a manipulator arm to find a site to dock it with the protein molecule. The manipulator arm allowed the chemist to feel the electrostatic forces and collision forces as the molecules interacted, as well as providing visual feedback. Audible clicks were also added to notify the chemist that a collision had occurred [BROOKS88]. This project, the predecessor to the DOCKER program used at the Wright Laboratories, touched on all of the areas that were outlined in Sutherland's vision as vital to "The Ultimate Display".

2.4 DOCKER at Wright Labs

2.4.1 Structure of DOCKER Program

DOCKER is a molecular docking program which was initially developed by the University of North Carolina, Chapel Hill. The program is designed to communicate with three servers to coordinate the docking task. The display server is currently run on a Silicon Graphics Inc. (SGI) graphics workstation and is responsible for drawing the molecular graphics and getting user input through an interactive menuing system. The ARM server is run on an Intel 80386 and is responsible for controlling the PER-Force force-reflective arm which is the interface allowing the chemist to provide input to move the drug molecule and receive output of the total force on the drug molecule. The final server is the energy server which is currently implemented on an Intel Paragon and is responsible for calculating the interactive forces between each atom and the total energy in the system.

2.4.2 Energy Model

The energy calculations are currently done using the AMBER force model developed by the Department of Pharmaceutical Chemistry at the University of California, San Francisco. This model was specifically designed to simulate proteins and amino acids. The model uses a relatively noncomplex calculation to determine the energy in the system taking into account the bond energy, bond angles, dihedral torsion, van der Waals forces, electrostatic forces, and hydrogen bonds [WEINER84].

The first term in the energy equation is a summation of all bond energy in the system. Bond energy is determined by the two atoms that are bonded together and the distance between them. Each pair of atoms that can occur in a protein or amino acid has a specific equilibrium distance and bond energy constant which are determined from empirical data. If the atoms are at their equilibrium distance, the bond contributes no energy to the system. If the atoms are not at their equilibrium distance, the bond will contribute an amount of energy to the system based on the difference between the actual distance and the equilibrium distance.

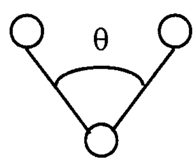

Figure 2-1: Molecular Bond Angle.

The second term is a summation of all bond angle energy. The bond angle is determined by three bound atoms (Figure 2-1). Similar to bond energy, each combination of atoms has a specific equilibrium angle and energy constant that are determined from empirical data. If the atoms are at their equilibrium angle, the energy contribution of the angle is zero. If the atoms are not at their equilibrium angle, the bond angle will contribute an amount of energy to the system based on the difference between the actual angle and the equilibrium angle.

The third term in the equation is the summation of the dihedral torsion in the system. Dihedral torsion is determined by the rotation angle of the center bond of four bonded atoms

(Figure 2-2). Again, there is an equilibrium position and energy constant which are determined from empirical data. If the dihedral is in the equilibrium position, it contributes nothing to the system's energy. If the atoms are not in the equilibrium position, the dihedral contributes an amount of energy to the system based on the difference between the actual position and the equilibrium position.

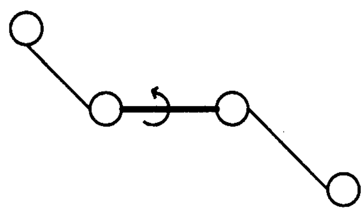

Figure 2-2: Dihedral Torsion or twisting of the center bond.

The fourth term in the energy equation is a combination of the van der Waals and electrostatic forces. Van der Waals forces are the weak attractive forces that exist between all atoms, even those with like charges. These forces increase as the atoms get closer to each other until a certain distance is reached and they become repulsive.

Electrostatic forces are the interactive forces between atoms based on their charges. This charge is usually due to a lack of electrons, yielding a positive charge, or an abundance of electrons, resulting in a negative charge. An atom can also have a partial charge if it is sharing electrons with other atoms.

The final term pays specific attention to hydrogen bonds. These bonds are not handled in any other term in the equation because they are very weak bonds which are easily created and broken. This volatility allows the chemist to fine tune the hydrogen bond distances and energies to desired values.

2.4.3 Modifications

The Wright Labs has made two modifications from the original UNC DOCKER program. The first is the replacement of the Argonne E-3 Remote Manipulator arm, added in the GROPE-III project, with a PER-Force force-reflective arm. The chemist uses this arm to manipulate the drug molecule and feel force interactions between the drug and protein molecules [LUPO95_1].

The second modification is the addition of the Intel Paragon. The energy server can be moved to the Paragon to run the energy calculations in parallel speeding up the calculations and allowing the system to handle larger molecules. The addition of the Paragon also provides the flexibility to replace the AMBER force model with a more complex force model in the future and to add additional functionality to the energy server.

2.5 The DOCKER Process

The process used to determine a docking position between two molecules with the DOCKER program involves three major stages: preprocessing, docking, and postprocessing [LUPO95_1, LUPO95_2]. The preprocessing stage (Figure 2-3) includes nine steps. The first step is to build the molecules in Cerius^2 or load molecules that have already been built. The chemist must then trim away parts of the molecules that will not affect the docking area to simplify subsequent energy calculations. The third step is to convert the molecular files to the DOCKER-readable .lg format with *msi2lg*. Next, the chemist moves both of the molecules to the origin in world space. Since the molecules are now overlapping in world space, the chemist must translate the molecules to correct their relative positions. In step six, the chemist creates a .gi file to define the extent of a grid encompassing the molecules in world space. The seventh step involves creating all of the remaining files necessary for DOCKER to run by executing the MD_ATOM and MP_LG scripts. This step takes on average 20 minutes to execute. Next, the chemist can load the molecules into DOCKER to see if the initial orientation leaves enough room for the exploration of the desired docking sites. If the extent of the grid leaves the protein backed into a corner or empty space on the wrong side of the protein, the chemist must exit the DOCKER program and repeat steps four, five, six, and seven. Once the orientation is acceptable, the chemist can proceed to the docking stage.

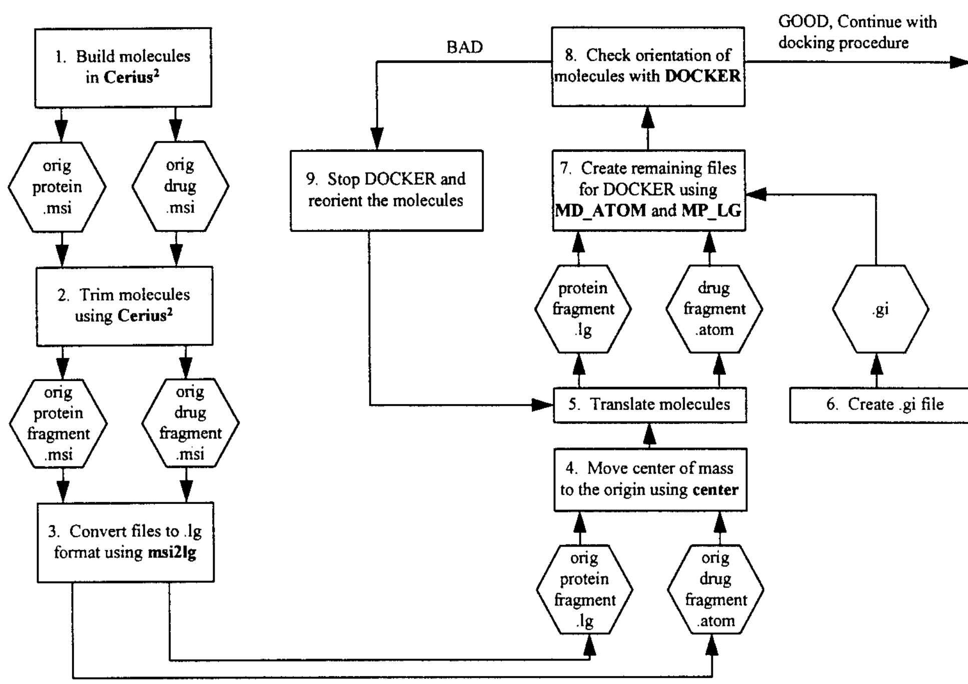

Figure 2-3: Preprocessing Steps for Running DOCKER.

The second stage of the process is the actual docking simulation (Figure 2-4). In step ten of the process, the chemist probes the protein molecule with the drug molecule to determine a desirable location for the drug to dock with the protein. Input to move the drug is provided by the chemist through the PER-Force arm. The docking process is aided by visual and haptic cues. The visual feedback tells the chemist about bond and interaction forces. The force cues, through the PER-Force arm, tell the chemist the approximate magnitude of repulsive, attractive, and torque forces present in the simulation. Once a desirable docking location is identified, the coordinates can be saved to a file.

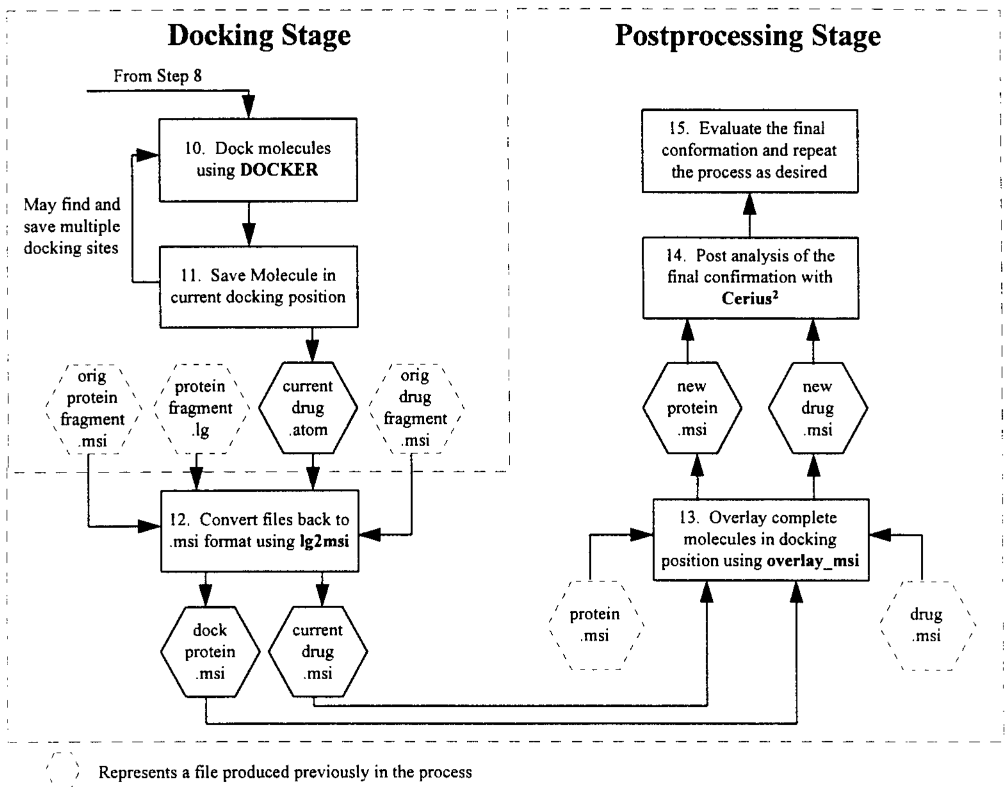

Figure 2-4: Docking and Postprocessing Steps for Running DOCKER.

The final stage of the DOCKER process is the postprocessing (Figure 2-4). Step 12 in the process involves converting the saved files from the DOCKER format to .msi format that Cerius^2 can read using *lg2msi*. This step requires the original protein and drug fragment files that were produced in step two. The *lg2msi* utility uses the same structure from the original file with the new coordinates from DOCKER. The next step is to overlay the fragment files obtained from DOCKER with the entire protein and drug molecules. The 14th step is to take the final confirmation, the complete protein and drug molecules that are in the configuration saved in the DOCKER program, and analyze it using Cerius^2. Based on this analysis, the final conformation is given a rating for comparison with other final conformations obtained from DOCKER.

2.6 Conclusion

The preceding summary of the current process has demonstrated that there are some problems in the process, such as the loop back in step nine, which waste the chemist's time and resources. A more detailed analysis of the problems in the process and an approach to address these problems will be presented in the next chapter.

DOCKER has a long history at UNC. Over the past 30 years several projects have been completed to extend the functionality of the program to the current product being used at the Wright Laboratories. The following chapter will include a detailed analysis of the current DOCKER program and an approach for implementing the modifications proposed in chapter one which are intended to extend the functionality of the program.

3. *Approach*

3.1 Analysis of the Problem

3.1.1 Orientation Tool

As demonstrated in Chapter 2, the ad-hoc orientation of the molecules in steps 4 and 5 (Figure 2-3) leads to the need for reorientation of the molecules after the chemist views them with DOCKER. The overall goal of the orientation tool is to save the chemist time in the preprocessing stage of the DOCKER process. It accomplishes this goal by replacing steps four, five, six and eight in the DOCKER process and virtually eliminating the possibility of looping back through step nine (Figure 3-1), ensuring that the user does not have to execute the scripts to build the display files in step seven more than one time.

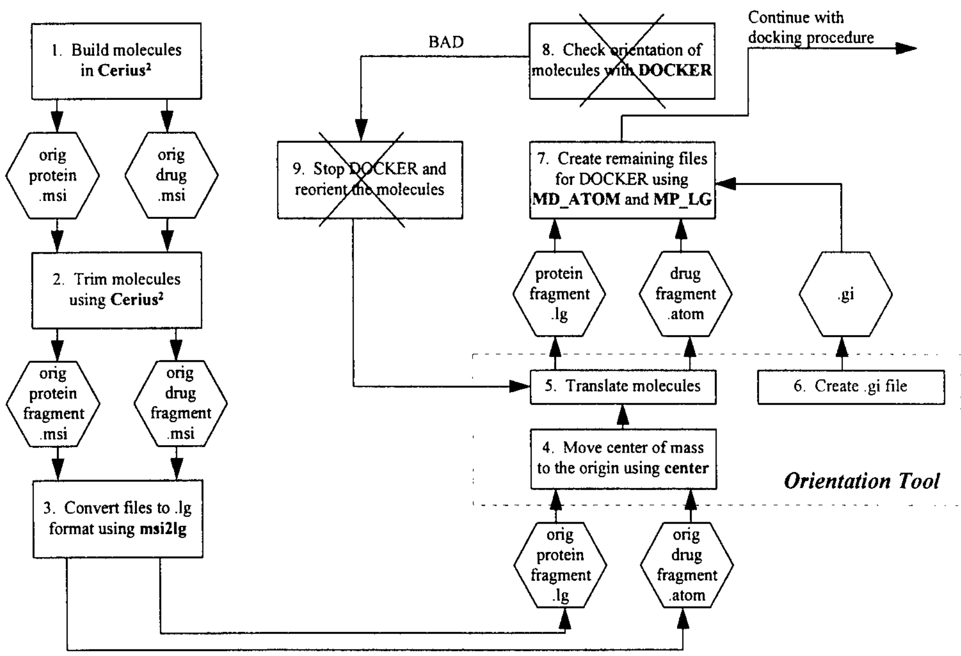

Figure 3-1: Preprocessing stage of the DOCKER process with the orientation tool implemented.

3.1.2 Articulation

The main problem with DOCKER is its lack of realism. This is due to the fact that DOCKER models the molecules as rigid objects. This is a simplification of the real-world interaction of molecules in which they flex. Articulation is a method for allowing flexibility of the molecules in DOCKER. It is achieved by allowing the atoms to independently move in response to external forces, such as other atoms or heat.

The addition of articulation improves the overall fidelity of DOCKER. This improvement is illustrated with an example of the bond angle energy from the AMBER force field equations used to calculate the interaction energy in the system (section 2.4.2). In a rigid model, the bond angle will never change. This means that the energy contributed by that angle will also never change. Conversely, in a flexible model, the bond angle can change. This motion will cause the bond angle to contribute different energy values to the overall interaction energy. This difference is then added to the other differences in each bond angle in the system. The other four components of the AMBER model will also react similarly illustrating the difference in interaction energy values between a rigid and flexible model. Knowing that the real-world interaction is flexible, we can see that adding flexibility to the model improves the overall fidelity of the model.

3.2 Analysis of DOCKER

The molecular docking process involves three stages: preprocessing, docking, and postprocessing. The analysis of the DOCKER system will focus on each of these stages individually.

3.2.1 Preprocessing Stage

The preprocessing stage involves creating the molecules, converting them to the appropriate format for DOCKER, orienting the molecules that are to be docked, and building the files necessary for DOCKER to display them. Appropriate tools are currently in place to create and convert the molecules and to build the display files; however, the orientation of the molecules is not supported. This orientation is done in an ad hoc manner that involves the user guessing the proper transformations to apply to the molecules without a visual representation. The lack of a proper tool usually forces the user to build the display files, a process which on average takes 20 minutes, view the orientation of the molecules with DOCKER, determine what the transformations should have been, and repeat the orientation and file building process with the correct transformations.

3.2.2 Docking Stage

The docking process is controlled by the *dock* program. To accomplish the docking task, *dock* uses three servers (Figure 3-2). The display server controls the presentation of the molecules and other visual cues, the energy server handles the calculation of the energy and force interaction in the system, and the arm server controls the input from and output to the PER-Force arm. Since the energy and arm servers are each implemented on different machines, *dock* must communicate with them over a network through message passing. The display server is implemented on the same machine, but it is a separate process, so it also communicates with *dock* using messages.

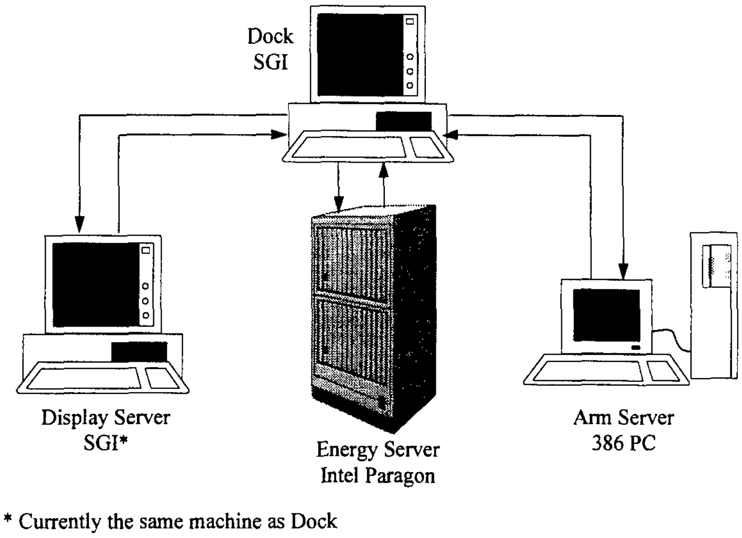

Figure 3-2: DOCKER System with the dock program controlling the energy, arm, and display servers.

3.2.2.1 Display Server Communication

The University of North Carolina, Chapel Hill, designed a display environment called the virtual world. It was intended to be a generic graphics system that could be used to display a multitude of objects. The display server uses the virtual world system to display the molecules and other visual cues in DOCKER (Figure 3-3).

The virtual world uses GL to display point, vector and polygon primitives. These primitives are organized into groups. Each group has properties associated with it such as a color to draw or a transformation matrix. Groups are organized into larger groups until the entire system to be drawn is in a single group. The draw functions then draw from the top group down.

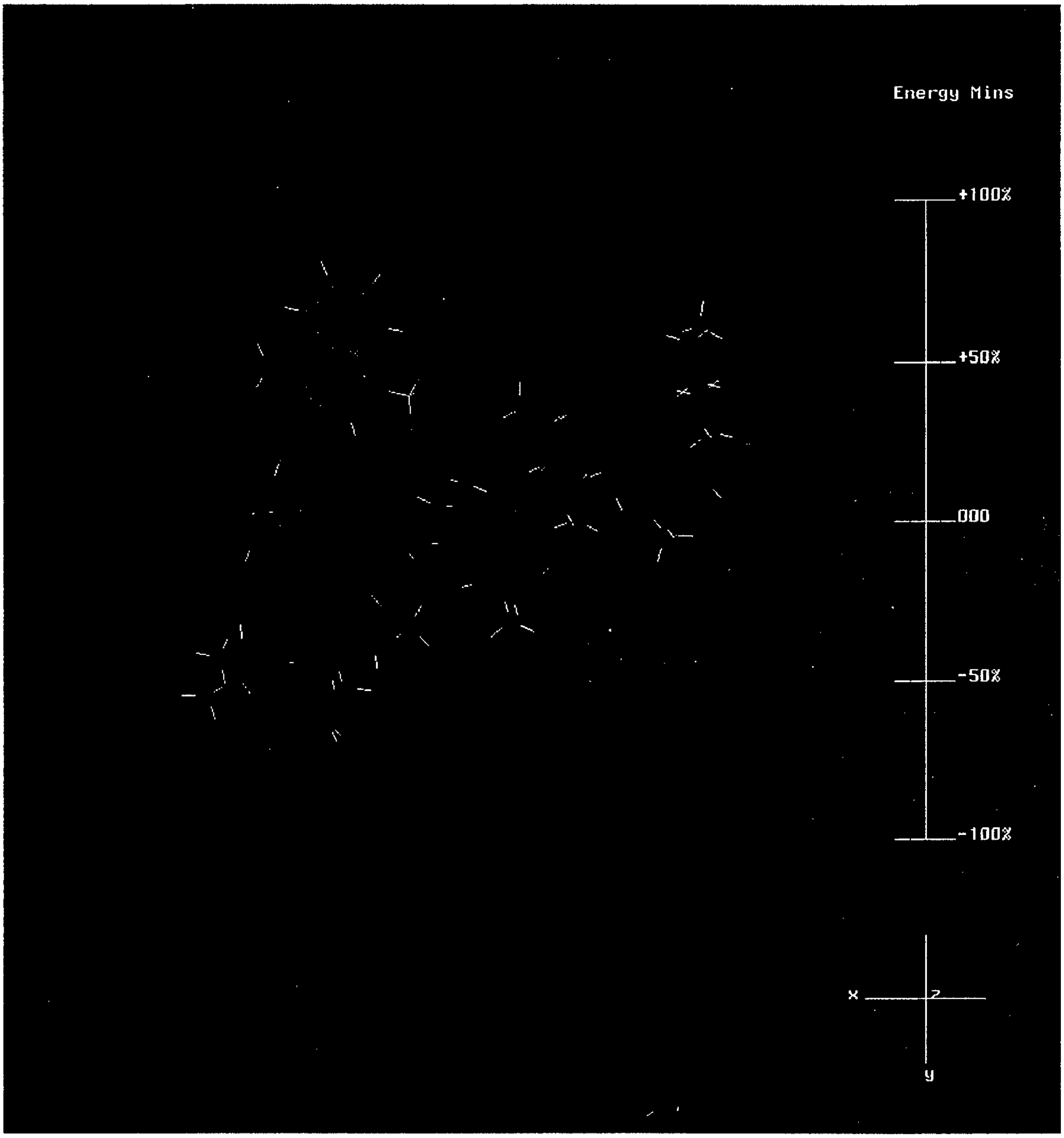

Figure 3-3: Example of the DOCKER display.

Dock builds its groups for the virtual world display by first dividing the molecules into a protein and drug group. These groups are subdivided across rotatable bonds, bonds about which a portion of the molecule can be rotated without rotating the portion on the other side of the bond (Figure 3-4). These groups are then subdivided by atom type. Dock then adds vectors to these subgroups representing the bonds that originate from the atoms in each subgroup. This structure makes it difficult to update the vectors individually when the locations of the atoms change because dock needs to know which group and subgroup the atom is located in and which vector in the list represents the bond that is intended to be updated. Currently, this information is known only by the preprocessing applications and the display server.

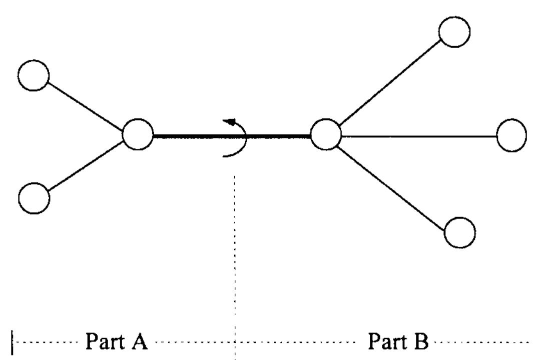

Figure 3-4: Rotatable Bond: Part A of the molecule can be rotated about the highlighted rotatable bond's axis without rotating part B of the molecule or vice versa.

When *dock* communicates with the display server it must use the virtual world structure to tell the server the location of each bond vector. Appendix A contains a description of the message codes that are used to accomplish this communication. An initial message is required to add the vectors to the display and subsequent messages can be used to update those vectors when the locations of the endpoint atoms have changed.

3.2.2.2 Energy Server Communication

The communications with the energy server are much less complex than those with the display server. *Dock* sends the energy server a message containing an integer code (Table 3-1) followed by a set of transformation matrices for the groups of the drug molecule.

Code	Function
0	Drop the connection to the energy server
1	Request energy between the protein and drug molecules and the drug and itself
2	Request the energy server to minimize the energy in the system (This function is not yet implemented in the energy server)

Table 3-1: List of the request codes that dock can send to the energy server.

The set of transformation matrices following the request code tell the energy server where the drug molecule is. The matrices have a transformation matrix for the entire drug molecule. They also have a transformation matrix for each rotatable group of atoms in the drug molecule. The product of these two matrices and the original coordinates of each atom gives the actual locations of the drug atoms during the docking process. The locations of the protein atoms are unchanged during docking since the protein molecule is static. The location for each atom is used by the energy server to calculate the energy in the system.

After a request is made to the energy server, *dock* calls a get function which reads the requested information from the energy server socket. This function waits until all of the requested information is received before allowing the program to continue. Once the data is received, a message is built to send to the display server updating the energy level display. The forces are then sent to the arm server to update the force reflection through the arm.

3.2.2.3 Arm Server Communication

Communications with the arm server consist of two requests. The first is a request for the current position of the PER-Force Arm. The arm server translates the input it receives from the arm joints into an orientation matrix and velocity vector in the arm frame of reference which is then returned to the calling program.

The second communication request is to update the forces on the arm. Dock sends this request to the arm server followed by force and moment vectors which represent the desired force on the arm based on the energy calculations. These vectors are translated by the arm server into commands to the arm joints.

3.2.3 Postprocessing Stage

Postprocessing is done by a separate tool, Cerius^2. This requires the user to save a docking position that he/she feels is a good position, exit the DOCKER program, load the molecules into Cerius^2, and run the analysis. The only information the user has when running DOCKER is the energy of the system and the force feedback to the arm. Providing the user with additional information about the final position of the molecules could save the time of storing and analyzing some undesirable docking positions.

3.3 Requirements for the Extensions to the DOCKER Program

3.3.1 Orientation Tool

I have designed an orientation tool to enable the chemist to orient the molecules with respect to one another. This tool has several requirements levied by the user. The first is to provide a visual interface to the user allowing him/her to put the molecules in the exact position desired before building the display files and loading them into the DOCKER program.

The second requirement is to provide the ability to manipulate the molecules using a mouse. The tool must provide the ability to individually translate and rotate each molecule and rotate both molecules together.

The tool must also provide a function to create a .gi file. This file defines the extent of the grid containing the molecules and some other information required by the dock program such as grid resolution and whether or not to include electrostatic and van der Waals forces.

The fourth requirement is to ensure the tool was interactive. The tool must have a very small delay between the action of the user and the reaction of the system.

The user must be able to display the tool on a SUN workstation as well as an SGI workstation. This is necessary because many of the chemists that will be using the tool have access to SUN workstations and need to be able to run the preprocessing steps remotely, setting up a DOCKER session on their workstation for lack of access to an SGI workstation.

The final requirement is to write the tool in C. This is because the maintainers of the tool are already familiar with C and they have the tools, compilers and libraries, to maintain it.

3.3.2 Articulation in Response to Interactive Atomic Forces

Articulation in response to the atomic forces allows each atom in both the drug and protein molecules to independently react to the forces applied to it by the other atoms in the system. My design of the articulation is governed by several user requirements, the first of which is to use the virtual world. DOCKER was designed to use the virtual world and there are several hooks throughout the code to the virtual world display system. The users were concerned that moving away from the virtual world would cause problems in the rest of the system.

The implementation of the articulation must not adversely affect the performance of the rest of the system. The users of DOCKER feel that any decrease in the current performance of the program will be unacceptable.

The third requirement is to provide several updates during the articulation process. Applying several updates allows the atoms to move in an animated fashion to their final positions. This provides the user some feedback that the process is proceeding and allows the user to determine if it is necessary to stop the process before the atoms reach their final positions.

Another requirement is to lock the drug molecule during the articulation process. This means that the user cannot manipulate the drug molecule while the system is articulating. This is necessary because the articulation will not be occurring in real-time so it will not be able to provide interactive results without slowing the entire system down and violating a previous performance requirement.

The fifth requirement is to write the modifications and additions to DOCKER in C. This is for the same reasons as the requirement for the orientation tool and because the rest of DOCKER is implemented in C.

The ultimate goal of the articulation is a minimum energy orientation of the molecules in their current positions with each atom in a position such that the sum of the forces on it are minimized. This information tells the chemist what the final molecule will look like with the protein and drug docked in that position. With this information, and the information previously provided by DOCKER, the chemist can determine if the molecule will have the desired properties of the needed material before analyzing it with a postprocessing tool.

3.4 Approach to the Design

3.4.1 Orientation Tool

The first design decision necessary was the selection of the language to build the tool in. The initial suggestion of the user was Motif, but Inventor was also considered. The decision was made to go with Motif for several reasons. The first was that the Inventor program had so much overhead, it performed very poorly on the systems the user was running. This poor performance violated the requirement to be interactive.

The second reason Motif was chosen over Inventor was the requirement to be able to display the tool on a SUN workstation. The SUN workstations used at the Wright Labs have the Motif libraries to support displaying a Motif program but can not support displaying an Inventor program.

Once Motif was chosen as the language, a decision had to be made about the design of the user interface. The user interface is intended to follow the standard windowing system design. Chapter four will present the specific details of the design of the user interface.

After reviewing the requirements that were provided by the user, I determined that there was a function that was overlooked: the ability to dolly the molecules back and forth. This function is necessary because there may be some molecules that will be large enough to extend beyond the edge of the display window. The user needs the ability to move his/her eye point further back so the entire molecule can be seen in the display window at one time. The user may also want to move in closer to the molecules for detailed work on smaller molecules.

3.4.2 Articulation in Response to Interactive Atomic Forces

Articulation allows the atoms in each molecule to react to the forces applied to it by the other atoms in the system. This requires the ability to modify the display to reflect the motion in the molecules. This brings up the question of how to modify the objects in the virtual world. My initial ideas included modifying the virtual world to allow modification. After researching the virtual world system, I was able to determine that it was already designed to allow the user to modify the objects in the display. With that structure in place, the display server need not be modified.

Knowing that the virtual world can modify an object with the appropriate message, the *dock* program must be modified to produce this message. To construct an update message for the display server, the program needs to know the structure of the objects. Specifically, *dock* needs to know which vectors are in each group. This structure is determined in the preprocessing stage of the DOCKER process. The decision I need to make is how to get this information. My initial idea was to reproduce the process that was accomplished in the preprocessing stage to reproduce the structure of the groups. Since the program would waste a lot of time rebuilding the structure, I have decided to focus on saving the initial structure from the preprocessing stage and passing it to *dock*. This is accomplished by writing the group structure to a file that *dock* can load and read when it must build an update message for the display server.

The next decision to be made is how to implement the articulation commands in dock. I have decided to implement the articulation as a loop within the main *dock* loop. There are three reasons for this decision. The first is to lock the user out during the articulation process. The articulation loop executes until the user enters a 'q' from the keyboard or the articulation is done. The user cannot manipulate the drug because the arm input is not read within the articulation loop. Once the loop is exited, the user can manipulate the drug again.

A second reason is to make it easier to do the articulation in several steps. The program will automatically repeat the articulation request without requiring input from the user. This meets the requirement to do the articulation in several updates.

The final reason for implementing the articulation in a loop is the requirement that the articulation not affect the performance of the rest of the system. Articulation will not be done until the user enters the articulation loop. If he/she does not enter the loop, the system will perform as it had before articulation was implemented.

Once the user has run the articulation, the atoms in each molecule will be in a new orientation. Since the chemists using DOCKER normally explore several docking positions in a single session to determine the best, he/she needs a way to return the atoms to their original orientation to explore another position. This is the reason that I have included a reset option in the main loop of *dock* with the articulation option.

3.5 Conclusion

This section has presented an analysis of the code in the DOCKER program and a summary of the requirements of the orientation tool and extensions added to DOCKER, and the approach taken to get to my initial design. The next chapter will discuss the details of the implementation of the orientation tool and the extensions to DOCKER.

4. Design

4.1 Orientation Tool

The orientation tool (Figure 4-1) is implemented in C and the user interface is constructed using Motif. The structure of the user interface is in figure 4-2. The Shell and MainWindow contain a Form widget which allows objects to be placed within it and attached to its sides. The Form widget contains a MenuBar which has two pulldown menu buttons and a help button. The DrawingArea provides an area to display and manipulate the molecules. The Label widgets provide areas to display information for the user. Finally, the Scale widget provides the dolly function to the user. These functions will be discussed in more detail in the following paragraphs.

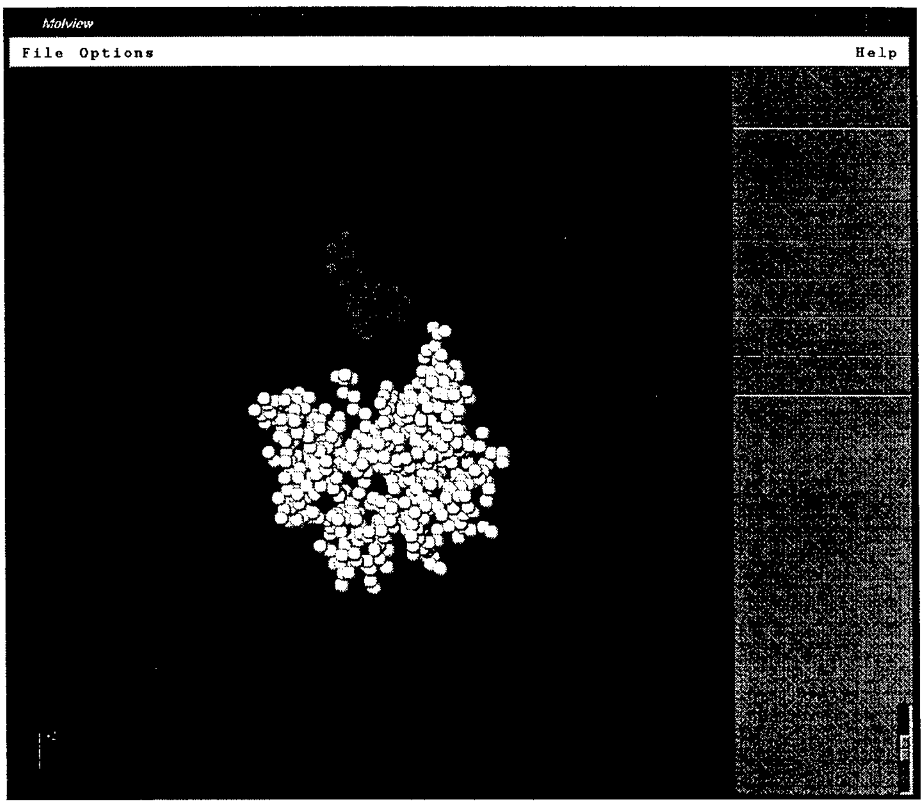

Figure 4-1: Orientation tool layout.

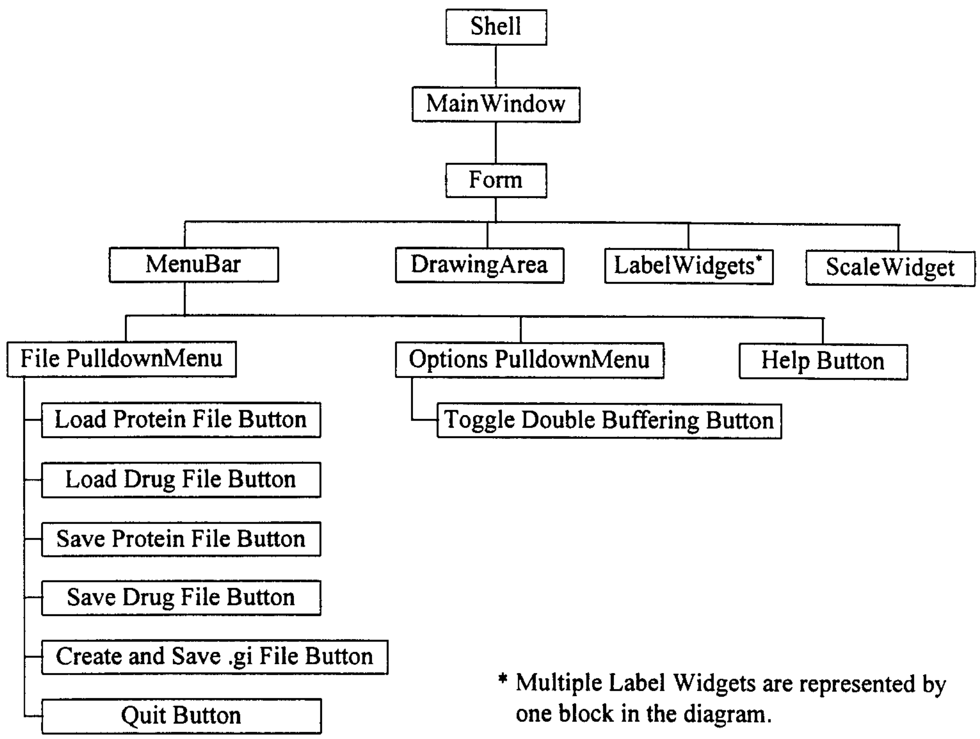

Figure 4-2: Structure of the Motif widgets in the orientation tool user interface.

4.1.1 MenuBar

4.1.1.1 File Menu

The design of the file menu follows standard windowing system conventions. It appears first in the MenuBar and contains standard functions to allow the user to load files, save files, and quit the program. These standard conventions make it easier for the user to learn and use the system since the functions are located in the same places as in other applications that the user is likely to be familiar with [NIELSON93: 28-29, 90-91, 132-133].

4.1.1.1.1 Load Files

The load buttons allow the user to select and load a protein and drug file. When the button is selected, a file selection window pops up allowing the user to search through his/her directories to find the correct files to load. The program allows the user to load a single molecule or both a drug and protein. If a protein/drug file is already loaded when a new protein/drug file is loaded, the new file replaces the old.

The load function expects that the files are in the standard large file, .lg, format. In this format each atom in the molecule has seven pieces of information: atom element name, x, y, and z-coordinates, partial charge, van der Waals radius, and atom type. This information is loaded into an array for use later in the program.

4.1.1.1.2 Save Files

The save buttons allow the user to save the protein and drug files individually. Pressing one of the save buttons pops up a file selection window allowing the user to specify the name of the file to save and its host directory. The file is saved in .lg format.

The same array of information that was read in the load function is written to the output file in the save function. The coordinates for each atom may have been modified with the manipulation functions that the program provides but the other information will be unchanged. This information will remain accurate because the program does not modify an atom's orientation with respect to the other atoms in the molecule which may affect the partial charge. It also does not change the atoms themselves which would affect the atom type and van der Waals radius fields.

4.1.1.1.3 Create and Save .gi File

The .gi file contains information for DOCKER about the extent of the two molecules in angstroms, the grid resolution, and some other information flags, such as whether or not to include electrostatic and van der Waals forces. Selecting this button pops up a file selection window allowing the user to specify the file name and directory the .gi file should be stored in. The extent of the molecules is determined from the actual minimum and maximum coordinate value along each axis. The remaining information is filled with default values as specified by the DOCKER users as the most often used values. The user can modify the default values in the file with another editor after creating it with this program if it is necessary for his/her simulation.

4.1.1.2 Options Menu

This menu contains the button that allows the user to toggle the double buffering on and off. When the program starts, the double buffering is on and the user has to toggle it if he/she wants it off.

4.1.2 DrawingArea

The DrawingArea is the window that the program draws the molecules in. It also provides functions for manipulation of the molecules.

4.1.2.1 Display

The program uses a perspective projection (Figure 4-3) to display the molecules in the DrawingArea. It also multiplies the screen coordinates by a scale factor because the

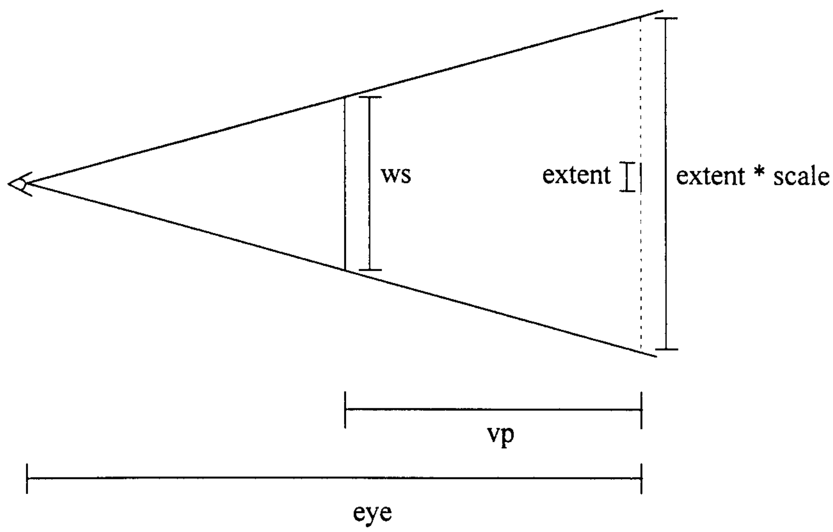

Figure 4-3: Perspective projection used to calculate the scale factor: ***ws*** *is the smaller of the height or width of the DrawingArea,* ***eye*** *is the distance from the object to the eye point,* ***extent*** *is the width of the square area to be displayed, and* ***vp*** *is the distance from the object to the viewing plane.*

extent of a typical large molecule will only be around +20 to -20 angstroms. Without the scaling factor, the user would be unable to see the details of the molecules. The scaling factor, as derived from figure 4-3, is

$$\frac{ws}{eye - vp} = \frac{extent \times scale}{eye} \tag{1}$$

$$scale = \frac{ws \times eye}{extent \times (eye - vp)}. \tag{2}$$

This scaling factor will expand the extent to a size that will completely fill the DrawingArea. This may leave the user little room to manipulate the molecules, so equation (2) is multiplied by an additional factor of one half to provide some extra space

$$scale = \frac{ws \times eye}{2.0 \times extent \times (eye - vp)}. \tag{3}$$

The program uses a right-handed coordinate system which is widely used by physicists [NIELSON93: 123-129]. The display plane is the x-z plane with positive x toward the right of the screen and positive z toward the top. The user is looking down the negative y-axis towards the origin.

4.1.2.2 Molecular Manipulation

The user manipulates the molecules in the DrawingArea with the mouse. Motif provides the ability to determine which mouse button is pushed, how far the mouse is dragged with the button pressed, and when the button is released. These functions are used to tie the translation and rotation functions to the mouse buttons.

The left mouse button translates a molecule. The user has to press the left mouse button over a molecule. The program then executes a callback function with information about where the mouse was on the screen when the button was pressed so the program can determine which molecule the mouse is over. If both molecules occupy the point the mouse was pressed over, the drug is selected. This selection is made because the drug molecule is almost always smaller than the protein. If the protein were always selected first and it completely overlapped the drug, there would be no way to select the drug without moving the protein. With the drug always being selected first, there will almost always be a portion of the protein molecule that is not overlapping the drug molecule to allow the user to select it.

After selecting the molecule, the user drags the mouse with the button held down to translate it. Another callback is executed when this motion occurs with information about where the mouse is. The difference between the previous location and the current location tells the program how many pixels the mouse has moved in the x and y directions. These translations scaled down by a factor of ten are used to translate the molecule in the display. The factor of ten slowed the motion on the screen enough to correlate the mouse motion with the molecular motion. Without it, the molecule would be translated one angstrom for each pixel the mouse moved on the screen.

The center button provides the rotation function for the molecule that is under the mouse when the button is pressed. With this function, the change in x and y position are used to create an angle of rotation, in radians, about the x and z axis respectively.

The right button on the mouse rotates both molecules in the same manner as the center button does for an individual molecule. If there is only one molecule loaded when this function is executed it only attempts to rotate the molecule that is loaded.

Each of these functions modifies the location of each atom in the molecule. The array that was loaded for each molecule in the load file step contains the coordinates of each atom. When one of the manipulation functions is executed on a molecule, the coordinates of each atom in this array are modified with the translation or rotation. When the user saves the molecule, the coordinates in the array for that molecule represent its current location in the display.

4.1.2.3 3D Effects

To create the illusion of three dimensions in the display the program uses Painter's algorithm. The atoms that are further back in the scene are drawn before the ones that are closer to the eye point. This makes it appear that the closer atoms are in front of the other atoms and becomes very noticeable when the molecules are translated or rotated.

Another method used to create the illusion of 3D is in the drawing of the atoms. The program draws two circles to represent each atom. The first circle is a darker circle centered on the x and y screen coordinates of the atom. The second circle is a lighter circle that is smaller than the first and is drawn on top of the first circle slightly off center. This gives the effect of a light source next to the user's eye point that is casting light on a sphere. This representation is not as visually pleasing as actually displaying the atoms as spheres but it is much less costly. Inventor was explored as an alternative for the display but was dismissed because it performed much too slowly. This was because the atoms were represented as actual spheres which contain several hundred polygons each. The effect was that the atoms appeared to be smooth 3D spheres, but the overhead of tracking all of the polygons and lighting effects proved to be unnecessary, as compared to the effect achieved with Motif, and unacceptable to the user.

4.1.3 Label Widgets

The label widgets display information on the current extent of the molecules along each axis and the volume of this extent in angstroms. Every time the program redraws the display, the values for the maximum and minimum extent along each axis and the volume are calculated and printed to the appropriate label. When the .gi file is created, the current extent values displayed in the labels are used to define the extent in the .gi file.

4.1.4 Scale Widget

The scale widget creates a slider to change the scale of the molecules that are displayed. When the user clicks on the slider and drags the mouse, the program executes a callback function that takes the current position of the slider and adds it to the extent of the display. As the extent gets larger, the molecules in the display are drawn smaller and vice versa.

The size of the circles drawn to represent the atoms also changes as the slider is moved. The size is determined by the ratio of the new scale factor to the original factor. As mentioned earlier, the scale factor is influenced by the extent of the display.

4.2 Articulation in Response to Interactive Atomic Forces in DOCKER

Articulation in DOCKER affects four major sections of the program: preprocessing tools, *dock*, the energy server, and the display server. The preprocessing tools are modified to provide *dock* with the structure of the vectors in the virtual world message format. The *dock* process does the bulk of the work to handle the articulation function. The energy server provides the updated coordinates necessary for the articulation. The display server does not require modification but supports a major portion of the articulation load. Appendix B provides a listing of the files that are affected by the implementation of articulation including a short description of the changes made to each.

4.2.1 Preprocessing

The tools *groups_to_vw* and *protein_to_vw* are used to create the virtual world messages that add the molecules to the display. These tools now build an articulation file in addition to the virtual world files (Figure 4-4).

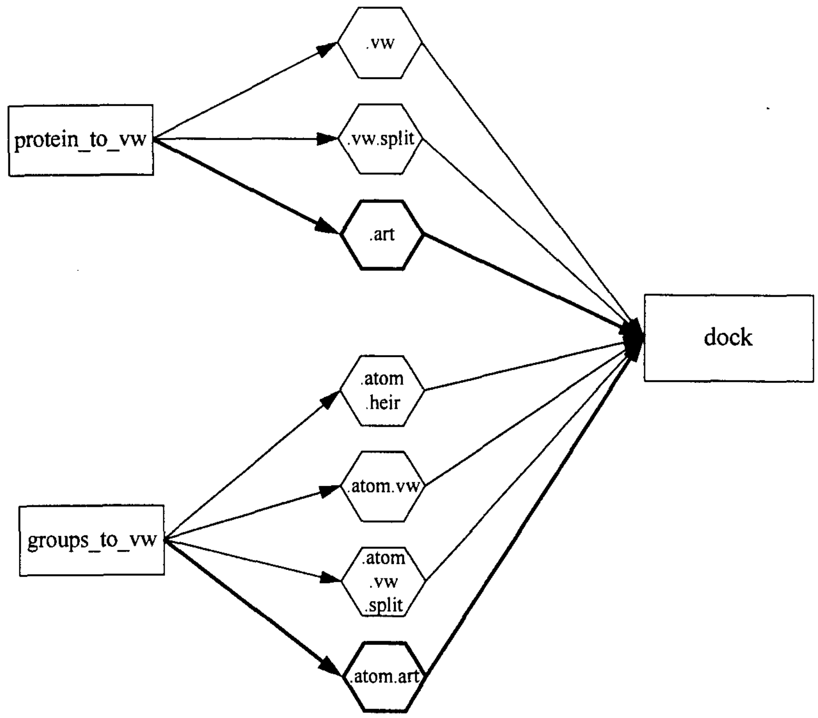

Figure 4-4: Files that are created from the protein_to_vw and groups_to_vw tools in the preprocessing stages of the DOCKER process with the addition of the articulation files highlighted.

The articulation file contains the group structure of each molecule. As the tools determine the group structure and the vectors contained in each group, they also write this information to the articulation file. When the tools are done, the virtual world file contains the message to add the vectors to the display and the articulation file contains the information necessary to update the vectors when they change.

The structure of the articulation file (Figure 4-5) is based on the virtual world message structure. Each group in the virtual world structure containing vectors is included in the articulation file followed by the source and destination atom number for each vector in the group. This allows *dock* to know which vectors are in each group when it needs to update the display. Each group is terminated with a sentinel of two negative ones. This sentinel was chosen because it is unique, since the atom numbers can never be negative.

51	←	Group Number 51
2 7	←	Vector from atom number 2 to atom number 7
3 8	←	Vector from atom number 3 to atom number 8
-1 -1	←	Sentinel Ending group 51
52	←	Group Number 52
7 2	←	Vector from atom number 7 to atom number 2
4 5	←	Vector from atom number 4 to atom number 5
1 5	←	Vector from atom number 1 to atom number 5
-1 -1	←	Sentinel Ending group 52
etc.		

Figure 4-5: Articulation File Structure

4.2.2 Dock

4.2.2.1 Articulation

The articulation option is the keyboard selection, 'a', in the main *dock* loop. When 'a' is selected the program enters an articulation loop that only exits when the articulation is complete, the user enters a 'q' from the keyboard, or an error occurs. The articulation loop performs three functions in each iteration: request an articulation update from the energy server, get the update from the energy server, and update the display with the atoms in the new positions.

4.2.2.1.1 Request Articulation Update

Request_articulation_update is a communication utility that creates and sends a message telling the energy server to do an articulation update. The message contains an eight-bit integer code to tell the energy server what to do. The code three is sent to indicate that *dock* would like an articulation update (Table 4-1).

Code	Function
0	Drop the connection to the energy server
1	Request energy between the protein and drug molecules and the drug and itself
2	Request the energy server to minimize the energy in the system (This function is not yet implemented in the energy server)
3	Request the energy server calculate the next articulation update
4	Request the energy server reset the coordinates of the molecules to the original orientation

Table 4-1: Updated list of the request codes that dock can send to the energy server including the articulation and reset options.

The code number is followed by a set of group transformations. These transformations contain the transformation for the entire drug molecule and the transformations for each group in the drug that is separated by a rotatable bond. These transformations, along with the coordinates for each atom which it already has in storage, allow the energy server to determine the actual location of each atom in world coordinates. These values are used to calculate the interactive forces on each atom.

4.2.2.1.2 Get Articulation Update

The *get* function reads the expected data from the energy server after the request has been made. This function expects to receive a 32-bit integer which represents the size of the data to follow. This value is calculated by the energy server and is based on the size of the drug and protein molecules. A size of zero indicates that the articulation is done and no information will follow.

The information following the size is the updated coordinates for the protein atoms followed by the updated coordinates for the drug atoms. Each set of updated coordinates is three floats representing the x, y, and z-coordinate for an atom. Each set of coordinates is copied into an array called atomlist which is then passed to the *update_display* function. *Get* waits until it has received all coordinates before it returns. This waiting is done with a loop that keeps reading from the specified socket until a given number of bytes of information have been read. If all of the information is read from the socket and the expected number of bytes have not yet been read, the loop will continue attempting to read until new information is placed on the socket.

4.2.2.1.3 Update The Display

Once the atomlist has been filled with the updated coordinates of the protein and drug molecules, the program builds the message to update the display. The first step in this process is to see if the display is ready. The program reads a flag from a second socket connected to the display server to see if it is writing a message to send back to dock. The display server needs to send a message to *dock* when running DOCKER in visual mode to notify it of the location of the drug molecule. *Dock* waits until the ready flag is sent. This check is skipped if we are using the PER-Force arm because the location of the drug molecule is sent to *dock* directly and is passed on to the display, so the display does not need to send a message back.

With the display server ready, the program builds the message to update the vectors in the display. First the protein articulation file is opened. For each group in the articulation file a message is constructed. Using the source and destination atom numbers for each vector, the program adds the appropriate codes (Appendix A) to the message for updating the vector with the new coordinates for those atoms. When the sentinel is reached, marking the end of the group, the program closes the message, sends it to the display server, and starts a message for the next group.

During implementation, I discovered a problem with this design. If a group contains too many vectors, the update message becomes so large it exceeds the message size limit of the display server and the system crashes. To correct this problem I have implemented a limit of 500 vectors per update message. When building the message, the program checks to see if the vector count is a multiple of 500, excluding zero. When this occurs the program closes the message, transmits it to the display server, and starts a new message to update the remaining vectors of the same group.

When the end of the protein articulation file is reached the drug articulation file is opened and the process is repeated until all of the vectors in the drug molecule have been updated. Finally, the articulation files are closed and the program checks for any user input from the keyboard and returns to the beginning of the articulation loop.

4.2.2.1.4 Structure of the Articulation Loop

The *request*, *get*, and *update* functions must be executed in order, but the program can plan ahead for the next update (Figure 4-6). The implementation of the articulation loop takes advantage of the fact that *request* starts a process on the energy server that can be executed in parallel with the *update* process in *dock*.

```
request articulation update
get articulation update
loop until done or keypress == 'q'
{
  request articulation update
  update display
  get articulation update
  keypress = get character from keyboard
}
update display
```

Figure 4-6: Structure of the calls in the articulation loop.

The loop starts by making a request for the first articulation update and waiting for it to return by calling *get* immediately. Once *get* has filled the atomlist with the first update, the program starts the energy server calculating the next articulation update with another request. While the energy server is running these calculations, *dock* runs the *update* function to build and send the update messages to the display for the current articulation update (Figure 4-7). As soon as the display is updated, the program gets the next articulation update and the loop is repeated until there is an error, the user enters a 'q', or the articulation is completed. After the loop is exited, the program updates the display with the last articulation update in the atomlist unless the articulation is done, in which case the atomlist will be empty.

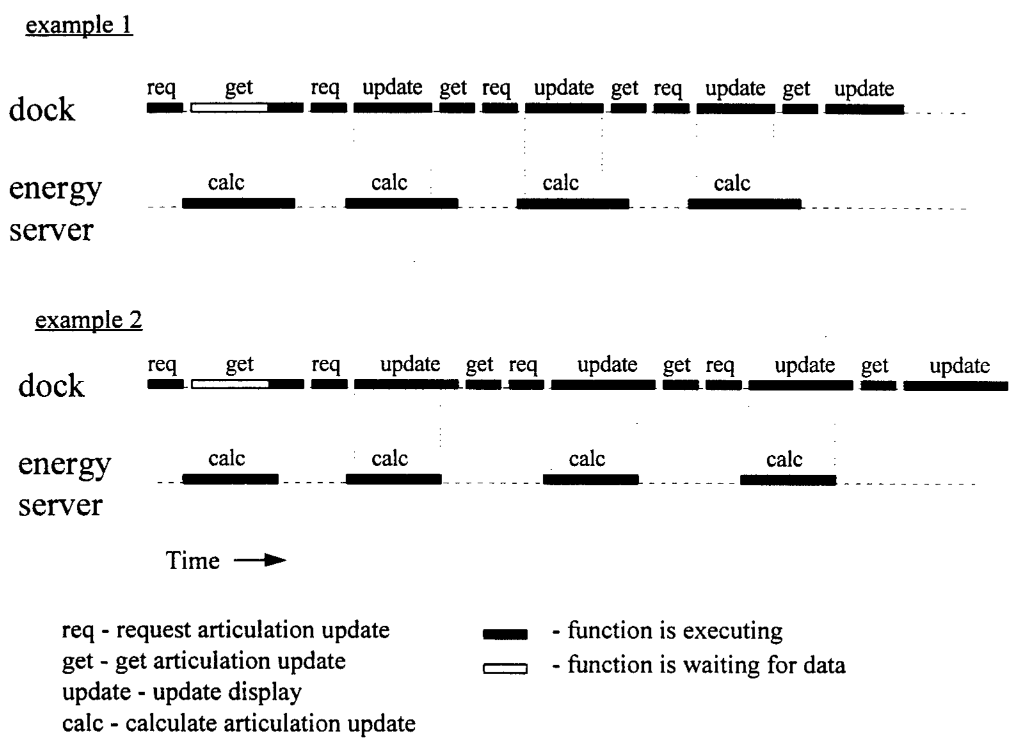

Figure 4-7: Timing chart showing the execution overlap in the articulation loop design between dock and the energy server. The first example shows the situation when the calculations take longer to execute than the update display function and the second example shows the case when the update function takes longer than the calculations.

4.2.2.1.5 Clearing the Energy Server Socket

The method used to get energy updates in the main *dock* loop is to make the energy request at the bottom of the loop, return to the top to handle the keyboard commands, including the articulation command, read the energy values from the energy server socket, and update the display. This means that when the user enters the articulation loop there is energy information waiting on the energy server socket. This information must be cleared so that the articulation routines can receive the necessary information from the energy server to support the articulation request. This problem is handled by reading these values out of the energy socket and throwing them away when the articulate option is entered. I decided to throw them away for two reasons. The first is that they will provide little to no useful information to the user. In the time between when the user stops manipulating the drug and he/she enters the keyboard command, the main *dock* loop will be executed at least once. This means that the energy values on the energy server socket will be very close to, if not exactly the same as, the energy values currently displayed.

The second reason is that these energy values will be meaningless after the articulation has occurred. The articulation changes the energy of the system and the user is only concerned about the new energy value until the system is reset to its original position.

Once the articulation is complete, the program makes a request to the energy server to replace the energy values that were thrown away with the new energy values. This is necessary because *dock* expects that the energy values are waiting on the energy server socket at this point in the main loop and executes the functions to get the energy values before it makes another request.

4.2.2.2 Reset Option

The reset option is also a keyboard selection, 'o', in the main dock loop. The letter 'o' was selected because the program is orienting the molecules in their original positions. The letter 'r' could not be used because it had another function associated with it. This problem highlights a larger problem with the user interface of the program in that single letter commands are used so often through the program that additional functions must use letters that are not representative of the functions that they perform.

The reset option returns the coordinates of each atom to its original coordinates. In *dock*, the original coordinates for the drug and protein molecules are stored in the arrays Drug_mol and Gridmol respectively. *Dock* copies the original coordinates into the atomlist array with the protein molecule followed by the drug molecule and calls the *update_display* function to send the update messages to the display server.

Dock also sends a message to the energy server. This message is an eight-bit code to tell the energy server to reset its current coordinate storage arrays with the original coordinates for the protein and drug molecules. The code for the reset option is four (Table 4-1).

The reset option does not require the program to clear the energy values from the energy server socket as is done in the articulate option. This is because the reset code does not need to receive any information from the energy server.

4.2.3 Energy Server

I modified the energy server to accept commands three and four to indicate a request for an articulation update and to reset the coordinates of the molecules' atoms, respectively (Table 4-1). The *articulation* and *reset* functions are implemented as stubs in this project and will need to be updated at a later date. This is because the energy calculations were outside of the scope of this problem as discussed in section 1.3. The intent was to demonstrate the ability of the display system to support the graphics necessary to perform the articulation and not to implement the actual calculations.

4.2.3.1 Articulation Stub

The *articulation* function is implemented as a stub for this thesis. The stub accesses a file to read one or more sets of updated coordinates that are returned when a request for an articulation update is made by *dock*. An input file called art.in is created with the correct number of coordinates for the protein and drug that is being displayed. When the end of the input file is reached, the stub returns the done signal as mentioned in section 4.2.2.1.2.

The *articulation* stub does not modify the internal storage arrays containing the coordinates of the protein and drug atoms. Because the energy calculations are outside the scope of this project, I did not want to create a storage structure that may or may not have supported the energy calculations that were to be implemented later. This means that when the articulation loop is exited, whether the molecules are in their original positions or modified positions, the energy values that are displayed will be unchanged. The actual articulation function, once implemented, will update the internal storage arrays with the new coordinates from the calculations and thus, the energy server will provide a new energy value for the system when queried for an energy update.

4.2.3.2 Reset Molecules Stub

The *reset* stub returns without doing anything. This is because *articulation* does not modify the internal storage arrays for the protein and drug molecules so the values that it contains are the original coordinates. The actual *reset_molecules* function will have to copy the original coordinates into the internal storage arrays to ensure that it contains the same set of coordinates as *dock* and the display server.

4.2.4 Display Server

The display server does not require modification to support the articulation. The virtual world was already designed with the ability to modify its objects, in this case the vectors. Appendix A, page A-3, has an example of a message that adds an object to the virtual world display and modifies another object. This is the method used to modify the molecules in the display.

4.3 Conclusion

This chapter has presented the design details for my implementation of the orientation tool and the articulation extensions to DOCKER. The orientation tool is implemented in Motif and provides a point-and-click interface that is similar to other windowing type applications.

The articulation extensions to DOCKER affect the preprocessing tools, *dock*, and the energy server. An important aspect of the design is the communication of information between the servers and *dock*. Articulation increases the message traffic between the servers significantly as will be presented in the next chapter.

The following chapter will discuss the results of my implementation. It will include some performance statistics and user feedback.

5. *Results*

5.1 Orientation Tool

Several tests were run to determine the quality of the orientation tool. These tests demonstrated that the requirements spelled out in Chapter Three were all met by the implementation described in Chapter Four. My fourth assumption discussed in section 1.2 was also proven correct because I was able to provide the manipulation desired by the user with Motif.

5.1.1 User Feedback

The user's responses to the orientation tool have been completely positive. They found the tool useful for orienting the molecules and its interface intuitive and easy to learn and use. These results were gathered from scientists who used the tool to support a project which required the simulation of docking several drug molecules with a protein molecule. The tool saved them the time of guessing the orientation of each pair of drug and protein molecules, building the files for DOCKER, and loading the molecules into DOCKER to view only to reorient the molecules to put them in their proper position, rebuild the DOCKER files, and reload them.

5.2 Articulation in Response to Interactive Atomic Forces in DOCKER

The implementation of articulation in DOCKER proved to meet all requirements discussed in Chapter Three. The first three assumptions presented in section 1.2 are also proven correct. Additional information collected on the articulation functions demonstrates the accuracy and performance of articulation in DOCKER.

5.2.1 Test Case Generation

With the exception of the water and carbon molecules discussed in the next section, all test cases were randomly generated. The process used was to start with the current orientation of the molecules, generate an arbitrary final orientation by adding a random value between negative one and one angstrom to each x, y, and z-coordinate. With the beginning and final orientation specified, I generated any number of steps between the two. These steps became the articulation input file, art.in, used by the energy server. Demonstrating that DOCKER can handle any arbitrary motion inputs during articulation also shows it can handle realistic articulation motion.

5.2.2 Articulation Accuracy

The accuracy of the implementation is demonstrated with a simple set of molecules: a water molecule consisting of three atoms and a carbon rod consisting of ten carbon atoms. The rod starts in a horizontal position with the water molecule above it. I calculated a series of steps to bend the rod up to enclose the water molecule and verified that the display reflected these events. This test demonstrates the articulation information gets from the energy server to *dock* and the update message created is accurate.

5.2.3 Articulation Performance

The performance information I am interested in is the performance of the articulation portion of the code. The articulation code consists of the functions *articulate*, *request_articulation_update*, *get_articulation_update*, and *update_display*.

Prof was used to collect the data necessary for this analysis. Prof is an operating system profiling tool, provided with UNIX, which identifies the number of CPU cycles used by each function.

5.2.3.1 Algorithm Complexity

Both functions *request* and *articulate* operate in constant time regardless of the number of atoms in the system (Figure 5-1). *Request* is constant because the message it builds for the energy server, consisting of the request code and drug transformation matrices, is always the same size. *Articulate* only includes the overhead of coordinating calls to *request*, *get*, and *update* and checking for user input from the keyboard.

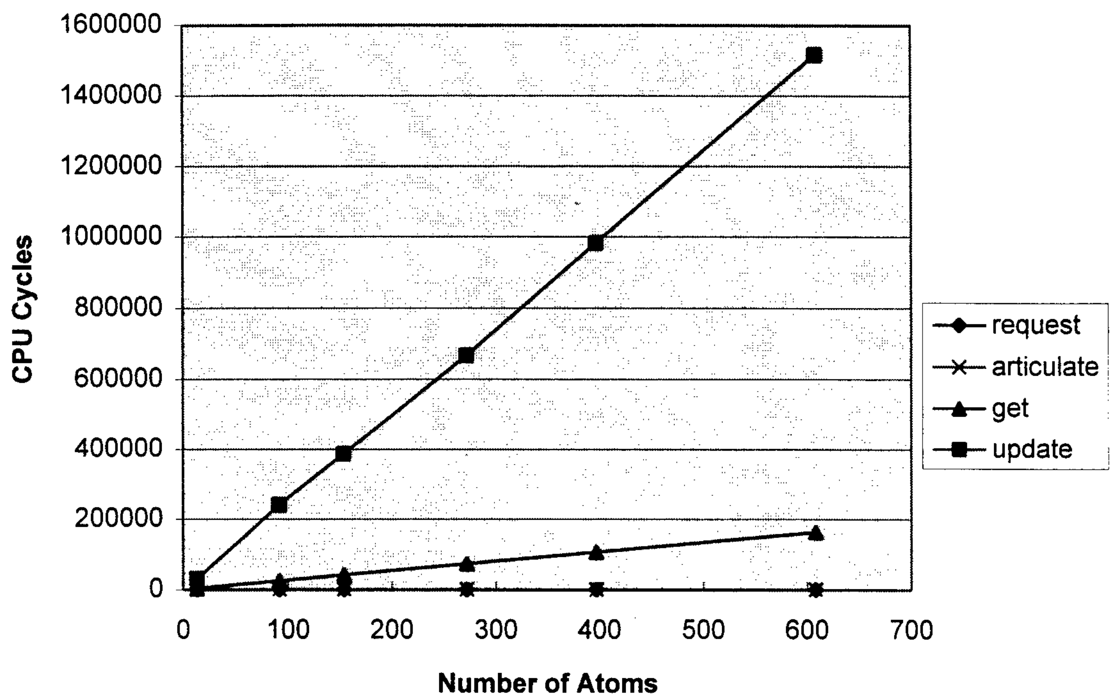

Figure 5-1: The number of CPU cycles used by the articulation functions for six test case consisting of 13, 92,154,272,397, and 608 atoms.

The *get* function performs in linear time proportional to the number of atoms in the system (Figure 5-1). Adding an atom to the system increases *get's* operating time by a constant amount (i.e. the time required to read the atom from the input file, transmit it to *dock*, and for *dock* to read the atom from the energy server socket).

The *update* function also performs in linear time (Figure 5-1). However, its performance is proportional to the number of bonds in the system. In the worst case, for the biological molecules that are used in DOCKER, an atom can have four bonds. Each bond is represented in the virtual world as two vectors. Each vector has six coordinates which is equivalent to two atoms. This means that the growth of *update* is proportional to 16 times the number of atoms in the system in the worst case.

5.2.3.2 Distribution of Work

It is important for maintainers of DOCKER to know which functions do the bulk of the work. As is seen in figure 5-1, the *update* function consistently uses more CPU cycles than the other articulation functions. *Update* takes on average 89% of the total cycles used by all of these functions (Figure 5-2). This information implies that any attempt to improve the performance of the articulation code within *dock* should focus on *update*. This conclusion, however, is based only on the performance of the current system with a stub for the energy calculations. The next section will discuss how the implementation of the energy calculations will change this conclusion.

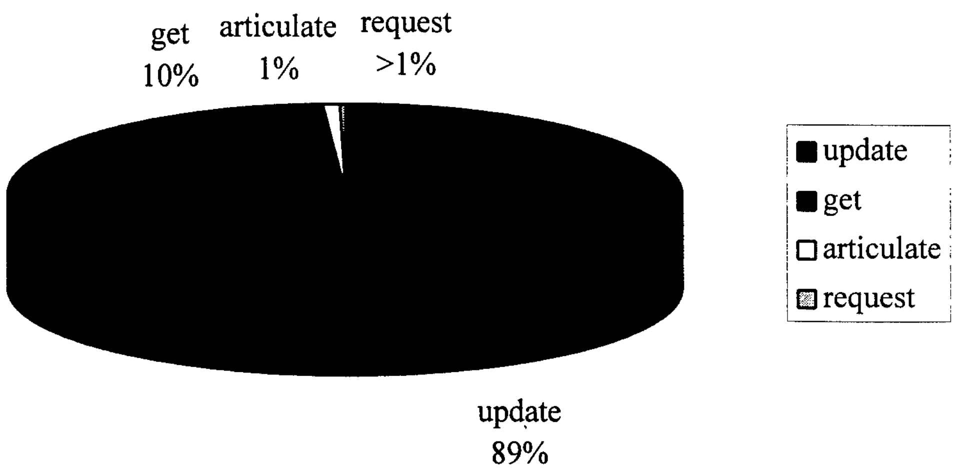

Figure 5-2: The percentage of the total number of CPU cycles in articulation code used by each individual articulation function.

5.2.3.3 Future Energy Server Implementations

Future implementations in the energy server will include calculations to determine new coordinates for each atom during articulation. This improvement requires a calculation to determine the mutual forces between atoms in the system. Thus, for n atoms in the system, $(n(n-1))/2$ forces must be calculated. If we assume each calculation takes some constant amount of time, then the overall order of complexity for this problem is $O(n^2)$.

Since energy server operations grow at a faster rate than for *update_display*, there is a point where the energy server calculations require more time to complete than *update*. At this point, the *get* function is affected because once *update* finishes *get* is called, but the energy server is not done calculating the new coordinates. *Get* must wait for the energy server to finish and then read the new coordinates. This waiting period is equivalent to the difference between the time taken by the energy server and the time taken by *update*. The difference will still be $O(n^2)$ in complexity which will cause the normally linear increase in the performance of *get* to become quadratic. This suggests that attempts to improve the performance of the articulation may need to focus on the energy calculations with secondary emphasis on the *update_display* function discussed earlier.

5.3 Conclusion

This chapter has presented the results of my implementation of articulation in DOCKER. These results have demonstrated that my initial assumptions discussed in Chapter One were valid and that all of the requirements presented in Chapter Three were met. The following chapter will identify future work that should be accomplished to extend and support the abilities of the orientation tool and DOCKER.

6. Conclusions

6.1 Future Suggestions for the Orientation Tool

The current implementation of the orientation tool has replaced steps four, five, six, eight, and nine in the DOCKER preprocessing stage. The next logical step is to include the file conversion process, step three, and the process to build the DOCKER files, step seven (Figure 6-1). This would simplify the overall DOCKER process because the user would only have to accomplish four major steps: create and trim the molecules in Cerius^2, use the orientation tool to convert and orient the molecules and create the input files for DOCKER, run DOCKER, and do the postprocessing analysis. Having the orientation tool perform the conversion and file creation functions saves the user having to remember all of the individual tools necessary to accomplish them.

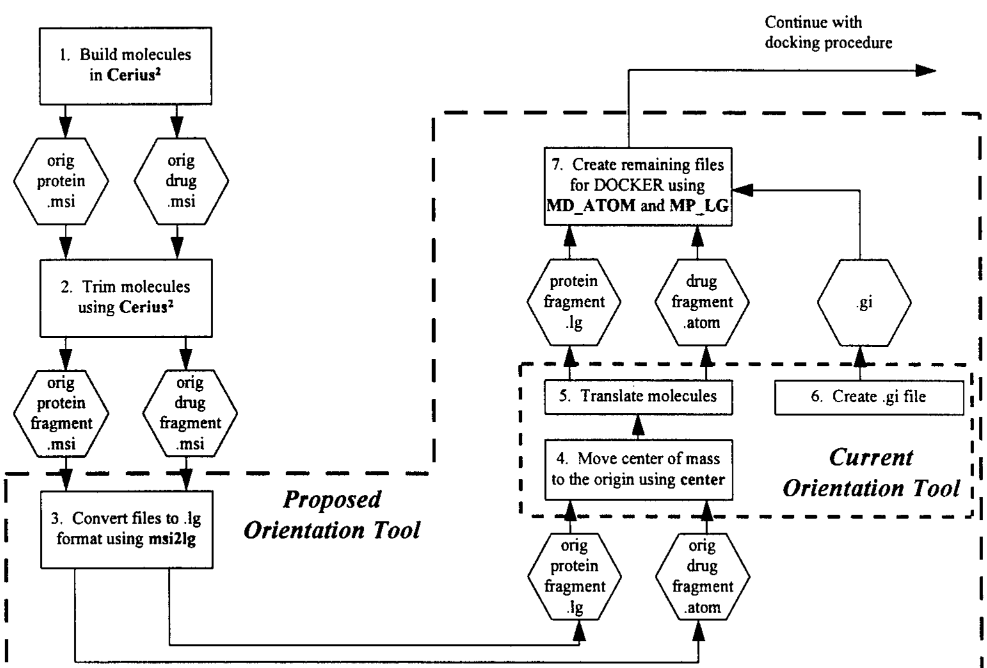

Figure 6-1: Preprocessing stage of the DOCKER process with the proposed orientation tool

The users' desires for this tool, as presented in section 1.5.3, also include functions to rotate a molecule about its center of mass, scoop a spherical region out of a molecule, and to make a planar cut through the molecules. Step 2 of the DOCKER process currently includes these functions (Figure 6-1). Adding these functions to the orientation tool would allow it to replace step 2 in the process, encompassing almost the entire preprocessing stage.

Once these modifications are implemented, it would benefit the users to be able to convert from any file format, not just Cerius^2's .msi format, to the .lg format DOCKER requires. This would provide users the freedom to use tools other than Cerius^2 to design, build and trim their molecules. There are tools currently available, such as Babel, which are designed to convert between molecular file formats and may be useful if incorporated into the orientation tool.

6.2 Future Suggestions for DOCKER

6.2.1 Energy Server

As mentioned earlier, the calculations necessary to compute the new coordinates for the articulation must still be implemented, as they are outside the scope of this project. There are two major problems with this implementation: the complexity of the calculations and the energy model necessary.

The complexity of a straight forward implementation of the energy calculations is $O(n^2)$ as discussed in section 5.2.2.3. This quadratic growth would suggest that alternate approaches to the problem should be explored. Research is currently being done at AFIT to predict the final orientation of the atoms in a protein molecule when it is folded using genetic algorithms

[KAISER96]. This approach could be used to predict the final coordinates of the atoms in the articulation problem as well as the intermediate positions.

The energy model currently used by DOCKER is the AMBER model [WEINER84]. The AMBER model, however, has two major disadvantages. The first is that it was only designed to simulate nucleic acids and proteins. This limits the molecules that can be simulated with DOCKER because the data tables only contain bond and interaction energy information for biologically legal combinations. Simulation of molecules that contain bonds or interactions that can not occur biologically is not possible with this model without a tremendous effort to extend the database.

The second disadvantage of the AMBER model is that the calculations are simplified. This simplification allows the model to perform but reduces its fidelity. This lack of fidelity is evident in a project completed during my research. Docking positions computed with DOCKER were analyzed by Cerius2 and the results did not match. This disagreement was determined to be a problem introduced by the simplifications in the DOCKER model.

The AMBER model should be replaced with the more robust CHARMm energy model [BROOKS83]. CHARMm is designed to handle bond and interaction energy calculations that are not included in AMBER allowing this model to handle the simulation of any molecules. It also provides more complex calculations to accurately model the interactive atomic forces necessary to support articulation. The implementation of the CHARMm model, along with articulation, will make DOCKER a more versatile simulation tool by improving its overall fidelity.

6.2.2 Virtual World

The virtual world system is designed to handle the manipulation of the molecules as a whole or in part, but in my opinion, it was not designed to handle the manipulation of the individual atoms. This is evident in the fact that *dock* must know the structure and order of the objects in each group to generate the update message. *Dock* must load additional files generated in the preprocessing stage to have access to the necessary knowledge.

Another indication that the virtual world was not designed to allow manipulation of individual atoms is the size of the required update message. Although the growth of the complexity in the creation and transmission of this message is linear, it increases by approximately 16 times the number of atoms in the system. The users of DOCKER have expressed the desire to view the molecules with methods other than the bond vector display which will increase the size of the message beyond 16 times. Options for display may include displaying a molecular surface, such as a Connolly surface [CONNOLLY83], a ball and stick diagram emphasizing the location of atoms as well as the bonds, a smooth ribbon representing the backbone of the molecule, and others. All of these display techniques require that several points be specified to define a surface. The virtual world would require that these points, possibly thousands to create a smooth and continuous surface, be included in an update message that would be transmitted to the display server for each update. This load would drive system performance to an unacceptable level for the user.

The two problems discussed here suggest that the virtual world may not be the best display tool for DOCKER. Although there are advantages to having a display that is independent of the domain being displayed, I feel it would benefit DOCKER to move to a molecular display system that has knowledge of the problem domain. Motif could be used to design this display. It provides the functions to draw points, lines, filled arcs, and filled polygons which could be used to draw any of the display options desired by the user. This implementation would require that the designer provide control for the grouping of atoms to allow each group to be manipulated individually. The advantage of this design is the reduction in the amount of information that is required to be communicated between *dock* and the display server. A change can be affected in the display by sending the new set of atomic coordinates and the display type desired by the user as opposed to the current virtual world message.

Another option for the display server is to use Inventor. Inventor provides hierarchical grouping of objects and the ability to display points, lines, polygons and custom designed objects. This option is listed second in my recommendations for the same reasons the orientation tool was not developed in Inventor (see section 3.4.1). Results obtained from the initial design of the orientation tool have suggested that an Inventor program does not perform well on the machine that the display server is currently implemented on. An improvement in hardware may provide the ability to support an Inventor display tool in the future, but the Motif design could be implemented now.

6.2.3 Articulation

The articulation implemented with this project accomplishes the first step in the progression desired by the users (see section 1.5.1), to have articulation at the users request. The next step was to update the energy model as discussed above. This leads us to their goal for real-time articulation, allowing the molecules to flex at all times including during user manipulation of the drug. Implementation of real-time articulation will require the interaction calculations be done quickly to avoid slowing the overall processing speed. The display system will also require modification as discussed above to simplify the messages necessary to effect a change in the display because the number of messages required will increase significantly. Even with this change, the overhead of articulation may slow the response time of the system to a point that will be unacceptable to the user as was mentioned in their requirements in Chapter Three.

6.2.4 User Interface

As mentioned in section 4.2.2.2, DOCKER's user interface leaves much to be desired. I would suggest a major redesign of the interface. The main problem is the long list of single letter keyboard commands that may or may not relate to the function that they perform, making it difficult for the user to remember. I would suggest that the user interface be a graphical interface consisting of pulldown menus with commands grouped by function, similar to that of the orientation tool. These menus could be activated with the mouse or the PER-Force arm. The arm would be the best option because it is already being used as the input device for the manipulation of the drug molecule. This would save the user having to change input devices to interact with the commands. Even if the user has to interface with the command menus using the mouse, this user interface will make it much easier for the users to learn and interact with the system.

This interface could be designed and implemented using Motif. All of the functions mentioned above can easily be supported as was demonstrated in the implementation of the orientation tool. This change would require that *dock* be replaced with the new user interface however. All of the functions that are performed with keyboard selections would become callbacks associated with functions in the pulldown menus. An interesting advantage that will come out of this proposed change is the calculation of the interaction energy in the system.

Currently, the energy is recalculated every time the dock loop is repeated whether the molecules have moved or not. Using Motif callbacks, the energy functions would only be called when the molecules are moved.

6.3 Conclusion

This thesis has made two major contributions to the DOCKER tool. The first is the implementation of the orientation tool. This tool has simplified the preprocessing stage of the DOCKER process. It provides the ability to visualize the molecules during the orientation process, eliminating the guessing that was done to orient the molecules previously. The result of the implementation of the tool is a reduction in the amount of time spent in the preprocessing stage of the DOCKER process.

The second contribution is the demonstration of the feasibility of adding articulation in response to interactive atomic forces to DOCKER. The results have shown that the program can handle the increase in message traffic to support the articulation at the users request making my initial assumption in section 1.2 correct. However, as presented in previous sections, I see problems with the current system being able to support multiple display options and constant real-time articulation. These are options that the users would like to see added to DOCKER in the future. As discussed in the previous sections, some modifications will need to be made, such as changing from the AMBER energy model to the CHARMm energy model and using a molecular display system rather that the virtual world, to allow the options to be supported in the future.

A. Virtual World Message Structure

To communicate with the virtual world display system the user must build a message packet using the virtual world structure defined in this appendix (extracted from *vw_packets* in the include directory) and transmit it to the socket that the display is connected to. The contents of the packet will specify the addition of a new object or modification of an existing object. The format of the packet is a sequential data stream organized as follows:

```
           +-- Header
           |  ID/data
Packet >--+  ID/data
           |  ...
           +-- ID/data
```

Contents	Description	Data type	Routine
HEADER	Number of chunks (64 bytes each)	vw_BLOCKNUM	read_packet
1	World info	vw_SELECTOR	read_packet
1	Object adding	vw_SELECTOR	Get_World_Info
ID	Object ID number (Required)	vw_OBJECT_ID	Collect_Object_Info
NAME	Object name (may be NULL)	vw_ASCII_STRING	"
1	Transformation mat (default I)	vw_SELECTOR	"
MATRIX		vw_UNC_XFORM	"
2	Visibility (default 1)	vw_SELECTOR	"
VISIBLE		vw_SELECTOR	"
3	Set default for later parts	vw_SELECTOR	"
1	Default Color (front-face)	vw_SELECTOR	Get_Default_Values
RED GREEN BLUE		vw_UNC_COLOR	"
2	Default Backface color (if any)	vw_SELECTOR	"
RED GREEN BLUE		vw_UNC_COLOR	"
0	End of defaults	vw_SELECTOR	"
4	UNC object (numbered as created)	vw_SELECTOR	Collect_Object_Info
1	ASCII label for this thing	vw_SELECTOR	Get_Object_Parts
LABEL		vw_UNC_LABEL	"
2	Point (not implemented?)	vw_SELECTOR	(Not implemented?)
POINT		vw_THREEVEC?	(Not implemented?)
3	Line (two points)	vw_SELECTOR	Get_Object_Parts
POINT1 POINT2		vw_SIXVEC	"
4	Triangle (Three pts, normals)	vw_SELECTOR	"
TRIANGLE		vw_UNC_TRIANGLE	"
5	(reserved)	vw_SELECTOR	"
0	End of parts description	vw_SELECTOR	"
0	End of one object build	vw_SELECTOR	Collect_Object_Info

2	Object modification info	vw_SELECTOR	Get_World_Info
ID	Object ID number	vw_OBJECT_ID	Do_Modify_Object
1	Transform Matrix	vw_SELECTOR	"
MATRIX		vw_UNC_XFORM	"
2	Visibility	vw_SELECTOR	"
VISIBLE		vw_SELECTOR	"
3	defaults (CAN NOT BE MODIFIED)	vw_SELECTOR	"
4	UNC Object type	vw_SELECTOR	"
1	An ASCII label in object	vw_SELECTOR	Modify_Object_Part
NUMBER		vw_UNC_PARTNO	"
LABEL		vw_UNC_LABEL	"
2	Point (not implemented?)	vw_SELECTOR	(not implemented)
NUMBER		vw_UNC_PARTNO?	"
POINT		vw_THREEVEC?	"
3	Line	vw_SELECTOR	Modify_Object_Part
NUMBER		vw_UNC_PARTNO	"
POINT1 POINT2		vw_SIXVEC	"
4	Triangle (not implemented?)	vw_SELECTOR	(not implemented)
NUMBER		vw_UNC_PARTNO	"
TRIANGLE		vw_UNC_TRIANGLE	"
5	(Reserved)	vw_SELECTOR	Modify_Object_Part
0	End of Part list	vw_SELECTOR	"
0	End of Object Modifications	vw_SELECTOR	Do_Modify_Object
4	Inter-object link modification	vw_SELECTOR	Get_World_Info
1	Add link	vw_SELECTOR	Do_Link
PARENT ID		vw_OBJECT_ID	"
CHILD ID		vw_OBJECT_ID	"
2	Remove link (not implemented?)	vw_SELECTOR	(not implemented)
PARENT ID		vw_OBJECT_ID	(not implemented)
CHILD ID		vw_OBJECT_ID	(not implemented)
0	End of link mod info	vw_SELECTOR	Do_Link
0	End of World info	vw_SELECTOR	Get_World_Info
4	Message	vw_SELECTOR	read_packet
1	Start sending position updates	vw_SELECTOR	Get_Message
2	Menu item	vw_SELECTOR	Get_Message
1	Create new menu	vw_SELECTOR	Get_Menu_Message
NAME		vw_ASCII_STRING	"
2	Add item to current menu	vw_SELECTOR	Get_Menu_Message
ITEM		vw_ASCII_STRING	"
RESPONSE		vw_ASCII_STRING	"
0	Done specifying menus	vw_SELECTOR	Get_Menu_Message
0	End of messages	vw_SELECTOR	Get_Message
0	End of ID/data information	vw_SELECTOR	read_packet

Several objects can be added or modified in a single message. An example of a message that defines a new object named 'hello' and modifies an object with the ID number of 3 follows:

1	vw_BLOCKNUM	Size of this packet in 64 byte chunks
1	vw_SELECTOR	World info
1	vw_SELECTOR	Object adding
1	vw_OBJECT_ID	value of this object's ID
'hello\0'	vw_ASCII_STRING	Name of the object (NULL-terminated ASCII)
3	vw_SELECTOR	Set defaults
1	vw_SELECTOR	Set default color
100 100 100	vw_UNC_COLOR	Default color = 100,100,100 (r,g,b)
0	vw_SELECTOR	End of default setting
4	vw_SELECTOR	Add part of the object
3	vw_SELECTOR	Add a line to the object
5 5 5 10 10 10	vw_SIXVEC	The vector to be displayed
0	vw_SELECTOR	End of parts description
2	vw_SELECTOR	Set visibility
1	vw_SELECTOR	Visibility is ON
0	vw_SELECTOR	End of add object
2	vw_SELECTOR	Object modification
3	vw_OBJECT_ID	value of this object's ID
4	vw_SELECTOR	Modify part of the object
3	vw_SELECTOR	Modify a line in the object
1	vw_UNC_PARTNO	Number of the line to modify
7 7 7 9 9 9	vw_SIXVEC	The new vector
0	vw_SELECTOR	End of parts list
0	vw_SELECTOR	End of object modification
0	vw_SELECTOR	End of World info
0	vw_SELECTOR	End of the packet

B. Modified Files Listing

This appendix contains a listing of the files that were modified/created to implement articulation in DOCKER. They are organized by the DOCKER directory in which they are located. A short description with each file summarizes the changes made to it.

B.1 Dock Directory

B.1.1 Action.c

This file is one of the two places that keyboard input is handled, the other is dock. Since both routines receive the input value, both functions must be prepared to handle a legal input value. This file was modified to handle the articulation, 'a', and reset, 'o', options. Since dock handles these options, this file only prints a message to give the user feedback to let him/her know that the keyboard input was received.

B.1.2 Articulate.c

This file was added to implement the articulation loop and the reset code. When dock receives an 'a' or 'o' it calls the articulate or reset_molecules functions in this file. The articulate function also calls the request_articulation_update and get_articulation_update functions in the file eservercom.c and the function update_display in the file dservercom.c.

B.1.3 Dock.c

The only modifications necessary were to implement the code to handle the 'a' and 'o' options from the keyboard. The code to handle the articulate option reads the energy server socket to clear the energy values that are queued from the last energy request. It then calls the articulation function in articulate.c. When this call returns, an energy request is sent to the energy server so the energy values that were cleared are replaced for the get function that is called later in dock.

The code to handle the reset option is only a call to the reset function in the file articulate.c. Since nothing is read from the energy socket, we need not clear the energy values as is done in the articulation section of the code.

B.1.4 Dservercom.c

The modification made to this file was the addition of the update_display function. This function takes the atomlist which contains the new coordinates for each atom in the protein and drug molecules and builds an update message, using the virtual world format, to modify the vectors in the display. The function uses the articulation files for each molecule that were build in the preprocessing stage by the files groups_to_vw.c and protein_to_vw.c to build this message.

B.1.5 Eservercom.c

This file was modified with the addition of two functions: request_articulation_update and get_articulation_update. The request function sends the request code for articulation, three, to the energy server followed by the transformation matrices for the drug molecule. The get function reads the updated coordinated from the energy server and puts them in the atomlist for use by the update_display function in the dservercom.c file.

B.2 Include Directory

B.2.1 Atomlist.h

This file was created by splitting the file molecules.h into two parts. This file now contains only the structure definitions. The reason for the split was that I needed to include the information now in atomlist.h and grid.h but there was a conflicting definition of ATOM between molecules.h and grid.h. The definition for ATOM was left in molecules.h and the information I needed was moved to atomlist.h and included with grid.h.

B.2.2 Molecules.h

This file was split into two pieces. This file contains the constant definitions and the atomlist.h file contains the structure definitions. The file atomlist.h is included to ensure that files including molecules.h get all of the information that they are expecting from the original design.

B.3 Paragon Directory

B.3.1 Articulation.c

This file was added to provide the articulation and reset function for the energy server. The articulation function is a stub that reads the new coordinates from the file art.in and returns them to dock. The reset stub does nothing but print a message to the log file.

B.3.2 Eserver.c

This file was modified to handle the codes for an articulation request, three, and a reset request, four. The main case statement was expanded to handle these codes and call the calc_articulation and reset_molecules functions in the articulation.c file.

B.3.3 Get_command.c

The only modification to this file was in the comments. They were modified to reflect the new options that are available to the user.

B.3.4 Protos.h

This file contains the prototypes for every function in the energy server. It was modified to contain the calc_articulation and reset_molecules functions which were added.

B.4 World Directory

B.4.1 Groups_to_vw.c

This file was modified to produce an articulation file for the drug molecule. The file contains the information necessary for dock to build an update message for the vectors in the display.

B.4.2 Protein_to_vw.c

This file was modified to produce an articulation file for the protein molecule.

Bibliography

[BATTER71] Batter, J. J., "GROPE-I: A Computer Display to the Sense of Feel", *Information Processing, Proceedings IFIP congress 71*, pp 759 - 763.

[BENNETT92] Bennett, David, et. al., *DOCKER, GRIP Team Project*, Department of Computer Science, University of North Carolina, Chapel Hill, January, 1992.

[BRITTON77] Britton, E. G., *A Methodology for Ergonomic Design of Interactive Computer Graphics Systems, as Applied to Crystallography*. Ph.D. Dissertation, University of North Carolina, Chapel Hill.

[BROOKS83] Brooks, B. R., et al., "CHARMm: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations", *Journal of Computational Chemistry*, 1983, Volume 4, Number 2, pp 187 - 217.

[BROOKS88] Brooks, Frederick P., Jr., "Grasping Reality Through Illusion: Interactive Graphics Serving Science", *CHI'88 Proceedings*, May 1988, pp 1 -11.

[BROOKS90] Brooks, Frederick P., Jr., et al., "Project GROPE - Haptic Displays for Scientific Visualization", *Computer Graphics*, August 1990, Volume 24, Number 4, pp 177 - 185.

[BRYSON96] Bryson, Steve, "Virtual Reality in Scientific Visualization", *Communications of the ACM*, May 1996, Volume 39, Number 5, pp 62 - 71.

[CONNOLLY83] Connolly, M. L., "Analytical Molecular Surface Calculation", *Journal of Applied Crystallography*, October 1983, Volume 16, Number 5, pp 548 - 558.

[HAYD95] Hayd, Helmut, et al., "GAME: A Computer Graphics Method for Calculating and Displaying the Molecular Electrostatic Potential", *Journal of Molecular Graphics*, February 1995, Volume 13, pp 2 - 9.

[HIMEI95] Himei, Hiroaki, et al., "Study of the Activity of Ga-ZSM-5 in the de-NO_x Process by a Combination of Quantum Chemistry, Molecular Dynamics, and Computer Graphics Methods", *Journal of Physical Chemistry*, 1995, Volume 99, Number 33, pp 12461 - 12465.

[ILLMAN94] Illman, Deborah L., "Researchers Make Progress in Applying Virtual Reality to Chemistry", *Chemical and Engineering News*, 21 March 1994, Volume 72, Number 12, pp 22 - 25.

[KAISER96] Kaiser, Charles E., Jr., *Refined Genetic Algorithms For Polypeptide Structure Prediction*. M.S. Thesis, AFIT/GCE/ENG/96D-13, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

[KILPATRICK76] Kilpatrick, P. J., *The Use of Kinesthetic Supplement in an Interactive System*, Ph.D. Dissertation, Computer Science Department, University of North Carolina, Chapel Hill, 1976.

[KUBO95] Kubo, Momoji, et al., "Mechanism of the formation of ultrafine gold particles on MgO(100) as investigated by molecular dynamics and computer graphics", *Applied Surface Science*, 1995, pp 131 - 139.

[LUPO95_1] Lupo, James A., "DOCKER - System Users Guide Version: 1.9", Wright Laboratories, Materials Directorate, Wright Patterson AFB, OH, 25 April 1995.

[LUPO95_2] Lupo, James A., "Special Advanced Studies for Hardened Materials: New Materials Design", Contract F33615-92-D-5000, Wright Laboratories, Materials Directorate, Wright Patterson AFB, OH, 30 September 1995.

[NIELSON93] Nielson, Jakob, Usability Engineering, Boston: AP Professionals, 1993.

[OUH-YOUNG90] Ouh-Young, Ming, *Force Display in Molecular Docking*, Ph.D. Dissertation, Computer Science Department, University of North Carolina, Chapel Hill, 1990.

[PALMER87] Palmer, T. C., *Docktool: A Dynamically Interactive Raster Graphics Visualization for the Molecular Docking Problem.* M.S. Thesis, University of North Carolina, Chapel Hill.

[STONE95] Stone, Richard, "A Molecular Approach to Cancer Risk", *Science*, 21 April 1995, Volume 268, Number 5209, pp 356 - 357.

[SURLES94] Surles, Mark C., et al., "Sculpting Proteins Interactively: Continual Energy Minimization Embedded In a Graphical Modeling System", *Protein Science*, 1994, Volume 3, pp 198 - 210.

[SUTHERLAND65] Sutherland, I. E., "The Ultimate Display", *Proceedings of IFIP 65*, 1965, Volume 2, pp 506 - 508, 582 - 583.

[VARSHNEY94] Varshney, Amitabh, et al., "Computing Smooth Molecular Surfaces", *IEEE Computer Graphics and Applications*, September 1994, pp 19 - 25.

[WEINER84] Weiner, Scott J., et al., "A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins", *Journal of the American Chemical Society*, 1984, Volume 106, Number 3, pp 765 - 784.

Vita

Capt. Todd R. Kellett [redacted]. He entered undergraduate studies at the University of New Hampshire in Durham, New Hampshire. He graduated with a Bachelor of Science degree in Computer Science in May 1991. He received his commission on 25 May 1991 through the ROTC program.

His first assignment was at Air Force Global Weather Central, Offutt AFB, Nebraska. He served as a software analyst and configuration manager. In June 1995, he entered the Graduate School of Engineering, Air Force Institute of Technology.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Dec 96	Master's Thesis

4. TITLE AND SUBTITLE
Molecular Articulation in Response to Interactive Atomic forces in DOCKER

5. FUNDING NUMBERS

6. AUTHOR(S)
Todd R. Kellett/Capt/USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2750 P Street
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/96D-15

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. James A. Lupo
Materials Directorate
Wright Laboratory, WL/MLPJ
Wright-Patterson AFB, OH 45433-7702

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Molecular docking aids in the design of materials supporting the current and future needs of the warfighter by simulating the real-world results of combining molecules. Specifically, it supports research and development in novel non-linear optical materials for laser-hardening and other advanced optical applications. Potential uses for these materials range from personnel and optical system laser protection to holographic information displays. DOCKER, our baseline docking system, minimizes the computational overhead of the simulation by modeling the molecules as rigid objects. This simplification can cause DOCKER's solutions to disagree with the real-world results, because molecules flex as they react to one another and to other external influences, such as heat or the presence of a solvent. Our research adds flexibility to the DOCKER model by articulating it in response to the interactive atomic forces. We anticipate this addition will improve the model's predictive capability by improving its overall fidelity. Articulation also provides a basis for other extensions. These extensions include thermal effects and computation of barrier energy along a reaction path.

14. SUBJECT TERMS
molecule molecule interactions, interactions, relaxation, low energy, intermolecular forces, molecular dynamics, molecular energy levels, dynamics, molecular properties, molecules, computer simulation

15. NUMBER OF PAGES
86

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements.***

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	**PR** - Project
G - Grant	**TA** - Task
PE - Program Element	**WU** - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with....; Trans. of....; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.
DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - Leave blank.
NTIS - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.